Task 6

Deliverable 5.0.5

FINAL REPORT

for

Contract DAKF11-96-P-0347

SBIR Topic A95-062

NETWORK SIMULATION OF TECHNICAL ARCHITECTURE

August 15, 1996

Submitted to:

Telecommunications Division

Information Science and Technology Directorate

Army Research Laboratory

Submitted By:

PREDICTION SYSTEMS, INC.
309 Morris Avenue
Spring Lake, NJ 07762
☎ (908)449-6800
🖹 (908)449-0897
💻 psi@village.ios.com

TABLE OF CONTENTS

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

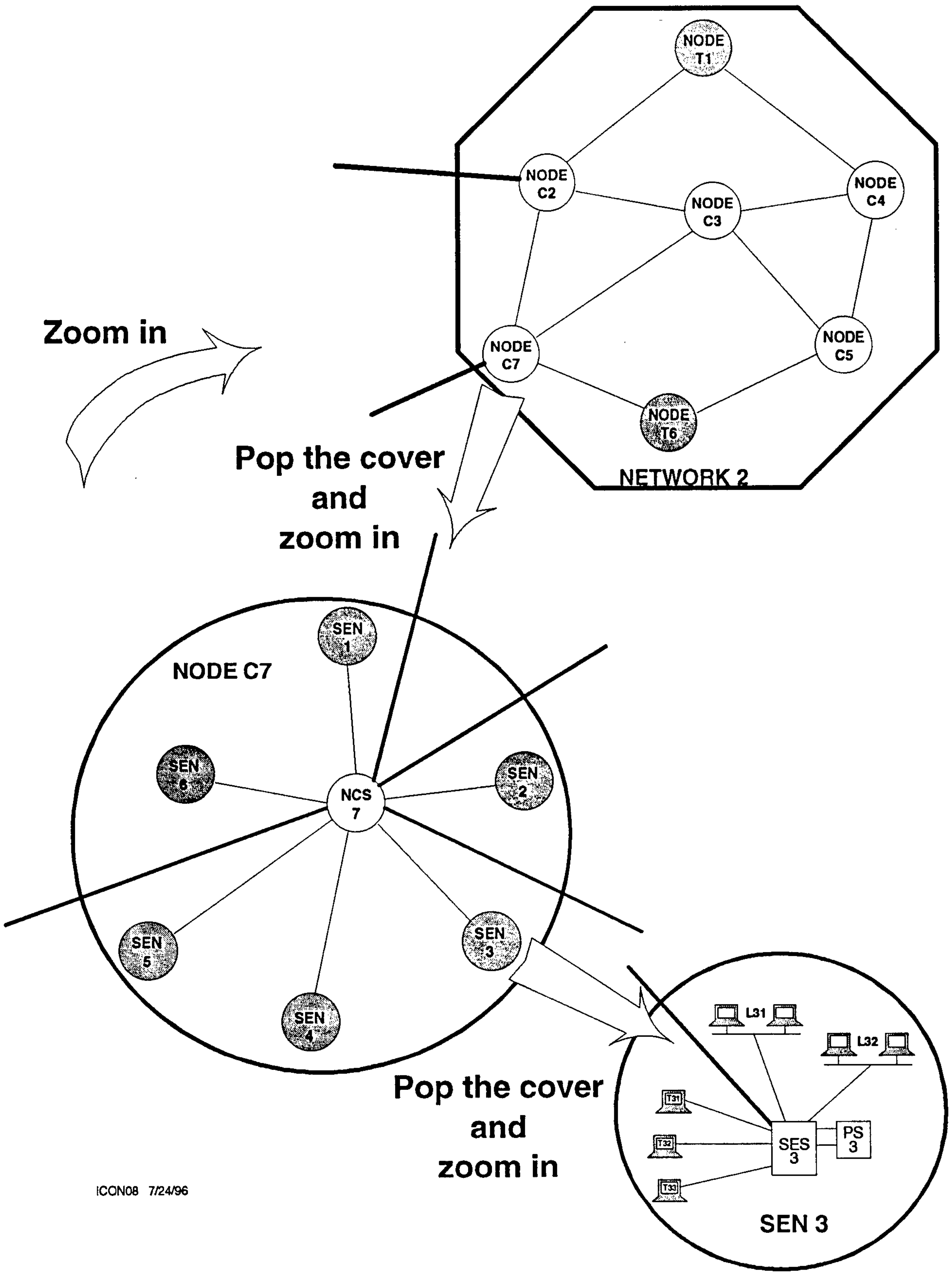

Figure 1-2. Popping the covers and zooming in on the details.

THE MOTIVATION FOR HIERARCHICAL ICONS

Why are hierarchies of icons important in simulations, particularly simulations of communication and control systems? Because the real entities being simulated are constructed of hierarchies of elements. To use an example, consider the communication network depicted in Figure 1-1. At the top of the figure, a simple three node network is shown consisting of three interconnected icons. These icons each represent a subnetwork themselves. To understand what is in NETWORK 2, we can pop the cover off of the NETWORK 2 icon, and see the subnetwork at the next layer.

When we first pop the cover, it is difficult to see the subnetwork comprising NETWORK 2. So in Figure 1-2, we show the effects of zooming in to have a closer look at the subnetwork. It contains 7 nodes, 5 central nodes (e.g., C2) and 2 tandem nodes (e.g., T1). Nodes C2 and C7 are connected to the other higher level networks outside the subnet.

Similarly, if we want to see what is in NODE C7, we can pop the cover off that node and zoom in to see the next layer down. It contains 6 small extension nodes (SENs) connected to a node central switch NCS 7.

Finally, if we pop the cover off of SEN 3, we see the subnetwork at that node. It contains a small extension switch (SES 3), a packet switch (PS 3), three terminals (T31, T32, and T33), and 2 local area networks (L31 and L32). Depending upon the simulation, this may be the bottom layer for iconic representation available to the user graphically. The icons at the lowest layer are elementary icons. Hierarchical icons are made up of elementary icons or lower level hierarchical icons.

Given that we can deploy icons representing the particular real elements or hierarchies of interconnected elements, then we can greatly simplify dealing with the complexity of large networks. For example, there are 7 elements in node SEN 3 represented by 7 icons. If this is the size of an average small extension node, and the other subnetworks are averages sizes, then the three icon network at the top of Figure 1-1 represents 1000 elements. since this simple network can easily be represented by a single icon, we can deploy a network of 1000 elements just by deploying a single icon - *while the simulation is running.*

We now have the option of looking at any of the detailed layers at any node or subnode. We need not look at 1000 icons to see the picture or any details of interest. This can greatly simplify the analysts job when working with realistic deployments in the field.

In addition to viewing the picture, we also want to interact with the network, changing it as we see fit, again - while the simulation is running. For example, we may want to move more terminals into SEN 3. We may want to take these terminals from SEN 2. If the models are designed properly, the simulation will proceed to represent what would happen if this were done in a real exercise. One could also disconnect and reconnect nodes at any layer in the hierarchy, while the simulation is running. When this happens, individual subnodes may have to be redeployed, just as they would in a real exercise.

CREATING AND INTERACTING WITH SIMULATIONS USING ICONIC MODELS

Just as we can pop covers and zoom into more detailed iconic representations of physical systems, we also want to be able to interconnect icons to build networks of new systems - while the simulation is running. The only requirements are that the icons be available to the simulation, and that the model invoked when a particular icon is deployed represents what the user expects.

PSI has already demonstrated this facility using the current RTG system for "flat" network structures. However, when an analyst has created a new network, say at the SEN level in Figure 1-2, he would like to cover up the details using a SEN 3 icon. Having created the desired SENs, one can then easily create a NODE C7 type network using the NCS and SEN icons. Similarly, one can create NETWORK 2 type subnetworks using the C and T NODE icons.

With this facility, higher level models can be created just by interconnecting icons - while the simulation is running. These models can also be stored away for future use, simply as a scenario. This is critical for saving time when dealing with complex networks, particularly when one must look at some results prior to accepting the physical network architecture. A good example is a network using radio transmission to connect nodes. In this situation, the analyst wants to obtain some form of connectivity before accepting the network as representative of the real world situation. This can be adjusted graphically until it is ready for use. Then the scenario can be stored for future use as an initial deployment at another time.

Neither the concept nor the capabilities described are limited to things that connect externally, e.g., switches, terminals, etc. There is no reason that protocol layers, buffers, and other more elementary components can't be combined using icons. One must simply design models that support the type of interactive interconnection described above. A modeler that designs models in this way can provide a library of iconically represented models to analysts and engineers that can be interconnected to build a higher level model graphically. Additional facilities that are yet to be described, e.g., pin connections, will assist in this process.

BREADTH OF APPLICATIONS

Although the examples shown are communication system oriented, there is no limitation on applications of GSS and RTG to other areas of dynamic simulation. In fact, models have been created for intelligence systems, including sensor tasking, objects and their motion, various sensors and platforms in motion; air defense systems, including high and low speed aircraft, radar, targets, and attrition; fire support systems, including observers, target detection and identification, command and control, weapon systems, kill measures, etc. These systems all require 3-D modeling, and the graphics facilities support 3-D visualization.

Because of the ability to interact with the simulation while it is running, and the ability to tie the simulation clock to the real-time clock (or some fraction or multiple thereof), these facilities can be used for live hardware-in-the-loop testing and specifically multiple simulations interacting as part of a Distributed Interactive Simulation (DIS) network experiment.

2. DESIGN REQUIREMENTS - ISSUES AND RATIONALE

Several design requirement issues were resolved during Phase I of this project. These are described below along with PSI's rationale for their resolution.

ICON HIERARCHIES

To allow users to build icons that are in one-to-one correspondence with models, the same hierarchical structure used for models, in general, is required for icons. Refer to Figure 2-1. As in the model hierarchy, *elementary* icon types, e.g., A and B, have no icons below them. *Hierarchical* icons, e.g., C and D, can contain either hierarchical or elementary icons.

ICON DEFINITION

Where and how the user will be allowed to define icon properties and change these properties as well as perform transformations must be specified. There is no requirement to allow users to draw new icons - on the fly - during a simulation. Icons can be changed and interchanged, but not created. Creation would cause misunderstandings in a simulation due to ambiguities, cause confusion during maintenance, and unduly burden the design with overhead. It is far easier for a user to have the desired icons already built in a library, with well defined properties, ready to use. These properties include orientation, scaling, line style, color, thickness, and blinking. These properties can also be changed dynamically while the simulation is running.

Referring to Figure 2-1, we must also provide for creation and modification of icon hierarchies in the icon draw utility as well as during a simulation. Therefore, we must be able to perform transformations and grouping of icons in this utility as well as bring in other library icons to create groups. This utility then becomes a hierarchical icon library management facility that supports creation, change and deletion of hierarchical icons. The icon draw facility is just one part of the library manager.

INSTANCED ICONS

Instances of icons must be supported just as they are currently in RTG. When we deploy copies of the elementary icons, we will need to denote the instances of those icons, since the initial state vector of each icon will be different, and we must be able to identify each of them. But now, instances can occur at any level in a hierarchy, and PSI's design philosophy for the instanced model version of GSS could be applied directly since, depending upon the model design, there could be a one-to-one mapping between icon instances and model instances. On the other hand, the icon hierarchy need not follow a predefined instance structure; it can be dynamic. A description of this facility is contained in Section 4: Designing For Icon Dynamics.

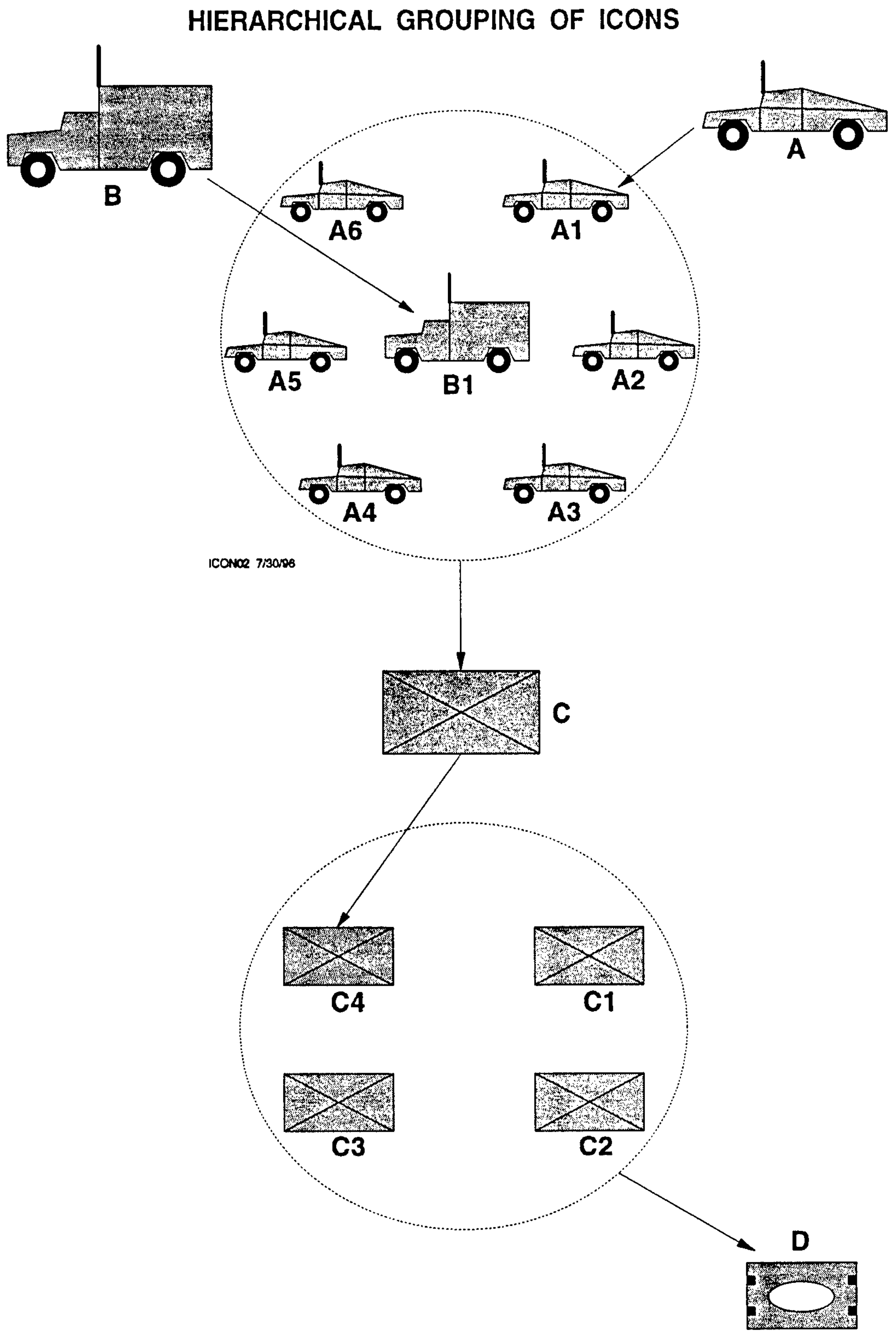

Figure 2-1. Hierarchical grouping of icons.

As an example of icon hierarchies, refer to Figure 2-1. If we deploy 5 instances of D, then we will have deployed 20 instances of C and therefore 20 instances of B and 120 instances of A. Each of these icons can correspond to a model, and PSI's design approach to model instances for efficient use of parallel processors can be applied directly to identification of the instances of these icons. This is explained briefly in the next section.

Future Design of Model Instances for Efficient Parallel Processor Utilization

PSI has briefed users on its plans for of an advanced version of GSS in which model instances would provide for efficient use of parallel processor machines. In the instanced model version of GSS, instances are defined at the model level as opposed to using quantities within a resource as is done now. All of the resources in an instanced model would be instanced, with each instance fully independent. CALL or SCHEDULE statements that reference processes inside an instanced model would specify the desired instance.

Since the resource instances used by that process are fully independent, and identified by the instance, subscripts need not be used within the process to refer to the instance that has being invoked, greatly simplifying the implementation. For example, C is a model with 4 instances. When a process in an instance of C calls another process in the same instance of C, then there is no need to specify the instance of C. If the called process is in an instance of A, one must specify the instance of A, since there are 5 instances corresponding to each instance of C. Calls to processes in B need not refer to B instances, since there are no multiple instances of B in C.

The major advantage of this design philosophy is that instances are associated at the model level, eliminating the need for explicit pointer references internally in the model specification, i.e., they need not be referenced inside a process. They can be handled implicitly behind the scenes by GSS. The point being made here is that this design philosophy is compatible with the hierarchical icon design presented proposed here. When PSI implements the parallel processor version of GSS, the hierarchical icon version of RTG will be supported.

RUN-TIME MANIPULATION OF ICONS

During a simulation, the user will want to interact with hierarchical icons graphically, pulling the cover off a grouped icon, and watching the individual icons within it as they move, change color, change shape, blink, etc. Users will also want to select a subhierarchy and change it. They may want to *detach* individual icons or subhierarchies from group. Changes to a group level icon will not affect a detached icon. Subhierarchies can be attached to other parts of a hierarchy. In PSI's RTG design, anything done interactively can be done within a GSS process. This facility has been preserved in the hierarchical icon design philosophy.

HIERARCHY OF ICON TRANSFORMATIONS

Modelers will want to perform transformations easily on their icons, e.g., a vehicle icon. Typically these would be position, and orientation. They may, on occasion, want to scale these icons based on simulation events. If position transformations (translations) are the only concern, e.g., when using nodes in a communication system, then the three real numbers (X, Y, Z) are sufficient. If changes in orientation of an icon are desired, then the Euler angles (θx, θy, θz) will be needed to describe rotation. If scaling is necessary, then scale factors (Sx, Sy, Sz) will also be needed.

For complex models and scenarios, modelers will want to group icons into hierarchies, perform icon transformations at higher levels in the hierarchy, and expect the icons in the lower levels to be transformed accordingly. For example, a turret on a tank must turn as a separate icon while sitting on a stationary tank. But if the tank is moving, it must move with the tank, keeping its orientation constant with respect to the tank when not moving separately. A group of tanks may move as a squadron, and turn as a squadron. Thus, all the transformations on each tank, and each turret on each tank, must be performed accordingly. All this must be done automatically when the highest level group icon is moved or rotated. Changes in position or rotations of subordinate icons can be applied as superpositions onto the hierarchical transformations, since all these transformations are linear.

If we store hierarchical icons in the library, we must be able to transform them from the icon library into a particular simulation as a *normal* or *initial view*, including, position, rotation and scaling.

Modelers will also want to be able to perform transformations on hierarchical icons at various levels in the hierarchy, while viewing them at different levels in the hierarchy. They will want to do this in the icon draw facility of the library manager as well.

Modelers will want to group a set of icons and put a cover on them with a new iconic identifier in the draw utility. They may want this new icon to *cover the area* covered by the original group as well as simply be a representative icon at their *center of mass.*

Modelers will want to pan, rotate, and zoom on scenes (changing the viewpoint) in a simulation, while hierarchical icons are undergoing their own transformations, independently, as a result of simulation actions.

USING GENERALIZED COORDINATES

Modelers do not want to think in terms of "pixel" coordinates. They want to deal in real numbers that represent the three dimensional space they are using to solve their problems, e.g., the rectangular (X, Y, Z) coordinate system, where X, Y, and Z take on positive and negative real numbers. Sometimes they may use the spherical (r, θ, φ), or cylindrical (r, θ, Z) coordinate systems. In these later cases, they will be willing to perform the translations to the rectangular coordinate system for graphics purposes, these transformations being simple and known.

Providing the modeler with generalized coordinate systems, as per the modeler's requirements for specific simulations, must be achieved. The modeler must be able to think of his problem in terms of his own coordinate systems, with minimal effort to perform the transformations.

SPEED AND ACCURACY ISSUES

Adding the features described above should not be done at unfavorable reductions in speed and accuracy of the existing facilities. In fact, by separating the modeler from the pixel coordinate system, and going to a real coordinate system, we should be able to gain accuracy of representations under the various transformations.

3-D ICON DESIGN

PSI has considered the ramifications of 3-D icons and has provided 3-D descriptions where appropriate. However, the real purpose of this project is to solve the hierarchical icon design problem to support deployment and motion of highly populated icon scenarios, where hierarchical grouping of icons can greatly ease the analyst's burden.

3-D icon design has been implemented in the data base portion of the Phase 1 design, but is not considered part of the Phase 1 design deliverables. This is because of lack of interest on the part of our communications system simulation clients, particularly CECOM. It is also because of the heavy overhead burden imposed by all of the ramifications of 3-D icons, including handling of other than triangular surfaces, closed surfaces, inside and outside surfaces, lighting, and all of the other aspects of polygons, particularly nonconvex, whose vertices do not reside in a plane.

Once the design for hierarchical icon transformations is complete, we can go back and assess the trade-offs of completing the 3-D icon design as part of Phase II.

SELECTION OF A STANDARD GRAPHICS LANGUAGE

As part of this effort, we have reviewed standard Graphics Languages (GLs) with a number of experienced graphics designers to determine what GL to use for the work proposed here. PSI's GSS-RTG currently uses the Silicon Graphics GL. This is a very powerful 3-D graphics language with special commands implemented only on SGI compatible graphics platforms. This is a limited subset of platforms, limiting portability of the system. When talking with SGI directly, we have been advised to use Open GL. It is compatible with SGI GL as well as X-Windows. Both can be called from Open GL extensions. In addition, Open GL has similar drawing philosophies and commands to the SGI GL. X-Windows is the most standard graphics library with UNIX. It is also used in Version 6 of GSS.

Since Open GL is fashioned after the SGI GL, it currently resides on all SGI platforms and SGI compatible graphics platforms, e.g., the IBM RS6000-PowerPC, as well as on other platforms and operating systems, including SUN 's SOLARIS. We understand that Open GL is being implemented on SCO UNIX, and Microsoft claims to be implementing it under its NT operating system.

Other "standard" GLs do not match the 3-D or vector draw facilities needed to perform hierarchical icon draws, particularly under the scaling or rotation transformations that are critical to the Army's scenarios. To the best of our knowledge, bit mapped icons do not fit the bill under current hardware designs that are oriented toward high-speed vector draws, including triangles and polygons.

Based upon our research, we have determined that the combination of X-windows and Open GL is clearly best for this and future projects so as to achieve maximum compatibility with other standard GLs and maximum portability across platforms.

3. BASIC DESIGN APPROACH

ACHIEVING GRAPHICAL INDEPENDENCE

To achieve graphical independence from simulation to simulation, a modeler will want to be able to scale his icons relative to each new simulation that uses these icons. For example, in one simulation, the world space may be 1000 kilometers by 1000 kilometers. In another it may be 50 kilometers by 80 kilometers. In each case, an icon must be scaled to the desired size in the world space so it can be seen on the screen as desired within a given *normal view* or intital projection. Then, when an icon appears on the screen in the normal view, i.e., prior to performing zoom functions, it will appear in size relative to other items shown on the screen. The facility to scale the library icon to the modeler's world space is provided in the GSS Simulation Control Specification as is currently done in RTG, where each icon is defined as part of the simulation.

In addition, modelers will want to transform icons, independently rotating, scaling, and translating them in their own coordinate system while the simulation is running. In this case, they will want to operate in their own relative coordinate system.

MODELER'S VIEW SPACE

Modeler's will want to lay out their graphics representation of simulation scenarios in a way that relates directly to their own physical coordinate system. For example, some modelers may want to think in terms of miles, others in terms of meters, and others in terms of angstroms. They should be able to perform their own transformations on their icons, and send the updated coordinates, orientation angles, and scale factors to RTG in terms of their own simulation coordinate system. In the Simulation Control Specification one defines their own world space coordinate system relative to the view space coordinate system used by RTG.

Consider that a modeler has created a coordinate system in two dimensions, (X, Y). Let's assume that miles were chosen for the coordinates. We will also assume that he has decided upon the size of the icons from a normal view, prior to zooming, for any given scenario. He must then define the size of each icon to be used in his normal view space in terms of miles. This value can be used in the Simulation Control Specification to automatically rescale the size of the icon from the library drawing scale to the normal view in the simulation.

BASIC DRAWING SPACE (BDS)

Referring to Figure 3-1, RTG icons are created with a special "drawing board" in Basic Drawing Space (BDS). This drawing space puts icons in the icon library in terms of pixels so they can be drawn on the screen. Thus, when creating an icon in RTG, one draws the icon in BDS using the drawing board facility. The icon is sized in terms of pixels when stored in the library. When stored, all vertices of an icon are put on the nearest pixel coordinate.

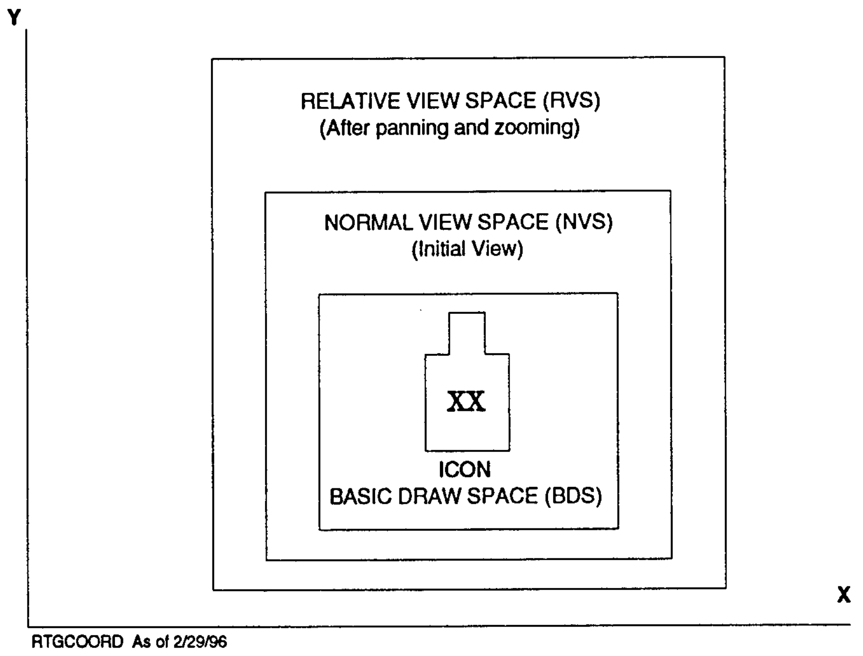

Figure 3-1. Modeler's View Spaces relative to ICON draws

In BDS, icons can be sized relative to one another. BDS allows the modeler to create library icons using the drawing board, and control the scale of the drawing board in each dimension (Xb, Yb, Zb) for each icon. When drawing or scaling icons for the library, BDS is not tied to the modeler's coordinate system in a particular simulation. It is strictly a library facility, where all scaling is relative to the library, and ultimately in terms of pixel counts. Therefore, the BDS space (Xb, Yb, Zb) is in pixel coordinates, and icons are stored in "relative" pixel coordinates using a BDS scale factor.

NORMAL VIEW SPACE (NVS)

The next consideration is that of viewing icons on the screen when running a particular simulation. In this case, the modeler wants to see icons in his view space at a given size relative to the "scene," *prior to zooming or rescaling.* We will refer to this as the *normal* or *initial* view. In the case where the scene is in miles by the modeler's scale, then the icons must be scaled to a size in miles to be displayed on the screen. For example, in a scene that depicts 20 miles across the screen when viewed normally (initially), then we expect to see icons scaled accordingly. For example, an icon may be 0.5 mile wide and 1.0 mile tall. Having all icons scaled automatically, relative to the "scene" or window view, is the prime consideration. The modeler need only enter the overall scale factor relating his own view to the BDS scale. In addition, the modeler may want to scale individual icons relative to BDS, and possibly scale them differently in different dimensions, e.g., X versus Y.

To accomplish this, the modeler will be required to scale his icons, relative to BDS, using the Normal View Space (NVS), (Xn, Yn, Zn) associated with the simulation. This can be accomplished using a single NVS - BDS scale factor, as well as by individual scale factors on each icon in NVS where different scaling is desired. All of this can be done in the simulation control specification for the initial state. These icons are then scaled automatically to their NVS scale factors during simulation initialization.

In RTG, a simple scene (e.g., a grid) could be drawn as an icon on the same drawing board as a very small icon using BDS. When viewing the simulation, a "very large" background icon is simply accomplished by a relative sizing of icons when viewed in NVS, as selected by the modeler.

RELATIVE VIEW SPACE (RVS)

The last consideration is the dynamic nature of graphics where the modeler is changing the position, orientation, and scale factors of icons relative to NVS while the simulation is running. These actions involve transformations on icons relative to the modeler's selected (NVS) coordinate system. When dealing with hierarchical icons, these transformations produce Relative View Space (RVS) in terms of (Xr, Yr, Zr) of the next higher level icon. A modeler will want to move icons around relative to the next higher icon level in a hierarchy. An icon in RVS can be considered an object that the modeler may transform relative to the next level group icon coordinate system. Of course, one can independently change the viewpoint by panning, rotating, and zooming the scene.

NVS-RVS IMPLEMENTATION CONSIDERATIONS

The principle concern of NVS is to accommodate the transformations from BDS to NVS automatically so that the modeler declares these once for a simulation. From then on, they are carried out directly as draw commands in the Graphics Language (GL), without explicit transformations. This should increase the speed of the draw.

In addition to the relative scaling of icons, we will want to build the hierarchical icon transformations so that they will be fast during run time. To do this, we can build the transformations between hierarchical levels so that when a transformation is performed at a particular level, the lower levels are transformed accordingly, as needed. It appears that no levels below that level being viewed on the screen should be transformed at that time. If these transformations are not done, then when a push down operation is encountered, all icons at the next level must be transformed at that time. Due to the nature of the hierarchical transformations, this should be a natural procedure. It should help to speed the redraws significantly since an operator or modeler may be working with transformations at a very high level most of the time, and never push to the next layer for a view.

If, during the course of a simulation, a transformation is carried out at a level higher than that being viewed, then the transformations to the level being viewed must be carried out at that time. When implementing this transformation, rather than going through a sequential set of transformations through the hierarchy, we must consider having a set of direct level-to-level transformations already computed in a reduced set of equations to speed up the operations.

HANDLING BACKGROUND OVERLAYS

Background overlays are supported in RTG with the User-Defined Background subsystem. Up to eight background overlays are supported. The major differences between background overlays and icons are:

- Icons are drawn by software automatically created by the icon library drawing board; background overlays are drawn by the modeler's own software.
- All background overlays are tied to the same NVS coordinates, being "registered" one to another, and moving together. Thus, the modeler can turn them on or off independently.

In the above sense, relative rotation and scaling of background overlays is handled the same way in NVS and RVS. Modelers must place the origin of the background overlay at a point with respect to NVS, just as an icon is placed by its origin. Orientation and scaling must be taken care of by the modeler in the same relative way. In general, pan and zoom operations are handled the same way as icons.

GENERALIZED RVS DRAW CONSIDERATIONS

Obviously, the drawing of background overlays and icons are imbedded in sets of draw commands that are called in a proper sequence. These draw commands are required whenever a graphical event occurs that requires a redraw of the screen. Without getting into the sequence of the draws, or the breakup of these draws to gain speed, we will consider the sequence of transformations to be performed to support hierarchical icon transformations in RVS under viewpoint transformations.

Icon transformations due to translation, rotation, and scaling would be performed first relative to NVS. These would be followed by viewpoint transformations of RVS.

When drawing highly populated dynamic icon scenarios, the nature of screen updates, and the timing of these draws is a major factor affecting speed of the graphics.

4. DESIGNING FOR ICON DYNAMICS

This section focuses on the design requirements for hierarchical icon dynamics at run-time. The design is described below along with PSI's design rationale.

RUN-TIME MANIPULATION OF ICONS

In communication network simulations, the number of icons, lines, and instruments can be huge (on the order of 1000's). These are the principal foreground graphic elements. During a simulation, users will want to insert, update, reshape, group, ungroup, and delete these entities. Some of these operations will require redrawing the complete foreground and some will not. Large foreground draws must be fast to keep up with the simulation. In addition, the Run-Time Graphics (RTG) system runs as a separate task. This will allow selected graphical inputs to occur without interrupting the simulation, a critical property for real-time simulations. Changes to the run-time icon database, to insert icons or change their properties, will be very fast compared to total foreground redraws. Therefore, foreground draws must be fast.

Changes to any of the foreground entities will result in an update of the database containing information on the state of the foreground elements, e.g., icons, lines, instruments, etc. This is followed by a draw of that element, or a redraw of the entire foreground scene. RTG currently separates background scenes from the foreground. This is because many communication system simulations involve radio transmission, and it is desirable to see placement of antennas on a map containing terrain contour lines and foliage, both important factors in determining propagation path loss and resulting network connectivity.

PSI has provided clients with antenna siting optimization simulations that allow users to interact graphically with the simulation while it is running. Users can deploy antennas on maps containing roadways, waterways, and towns, as well as terrain contours and foliage. They can draw boundaries that define mathematical constraints used by algorithms that automate optimal antenna placement. These are very complex scenes that take considerable time to draw compared to the foreground. These are best treated as background overlays and preserved in separate graphic bitplanes. These background scenes only need to be redrawn when panning or zooming.

DYNAMIC MANIPULATION OF HIERARCHICAL ICONS

When dealing with communication system deployments with large numbers of icons, users will reduce the screen clutter by representing naturally grouped icons (for example, those at a node) as a single icon. Deployment of multiple radios or nodes in a single company or battalion can be tucked behind a single icon to view only the interconnection of multiple companies or battalions. Groups of these hierarchical icons can then be tucked under another single icon representation, allowing the hierarchy to grow. When a user deploys the top level icon in the hierarchy, all sub-hierarchies and corresponding elementary icons will get deployed.

Figure 4-1 shows a typical hierarchy of a four layer hierarchical icon. E-ICONs are elementary and H-ICONs are hierarchical. Icons B(1), B(2), etc. are elementary instances of icon B. Q(1) and Q(2) are hierarchical instances of icon Q. Note that elementary icon E is connected to a higher layer in the hierarchy.

During the course of a simulation, users will want to move icons around with the mouse. They will want to do this at any level in the hierarchy. This implies that an icon at any level can be moved independently. If it is a hierarchical icon, then its sub-hierarchy will be moved accordingly. If it is an elementary icon, then it will be the only icon moved.

Users will also want to detach icons from their hierarchy. This implies that the icon is no longer part of the hierarchy, but is now a separate icon. One can also detach a hierarchical icon, in which case the sub-hierarchy remains intact but addressed independently.

Conversely, one will want to attach an icon to a hierarchy, in which case the attached icon becomes part of the hierarchy. Again, the attached icon may be a hierarchical icon, in which case all of the sub-hierarchies become part of the hierarchy. The user will want to attach and detach icons at any level in the hierarchy, with the exception that icons can only be attached to hierarchical icons.

CONNECTING AND DISCONNECTING ICONS

During the course of a simulation, users will want to connect two icons together to show a communication link, path, or some general form of connection between the two entities represented by icons. In the current RTG system, this is done by drawing a connect line between the two icons. The connect lines are drawn to the centers, i.e., the Basic Draw Space (BDS) [0, 0] points, of each icon. This can remain as a default.

With hierarchical icons, as shown in Figure 4-2, the user can build up a network of networks. This can be done by covering a subnetwork with a cover, e.g., covering the six instances of icon A and the single instance of icon B with the single icon C. When doing this, connections to the specific icons inside C must be known so that connections coming from the outside into C will be to their correct positions. Likewise, D covers the four instances of icon C with connections to the outside for use in a higher order network. To cover these subnetworks and preserve the correct connections to the outside, pin connections must be attached explicitly to the icons. Icons with specified pin connections are termed *pinned-icons.*

With the pinned-icon facility, one can deploy instances of the D icon knowing that each of these represents four instances of C icons that are specially interconnected and thus 24 instances of the A icons that are also specially interconnected. A complete network deployment can therefore be represented by a single icon. The ability to use a single icon to represent a complete deployment with special interconnections will cut the time to build scenarios. The ability to create and save such scenarios interactively, while the simulation is running, will be a major breakthrough for communication system simulation.

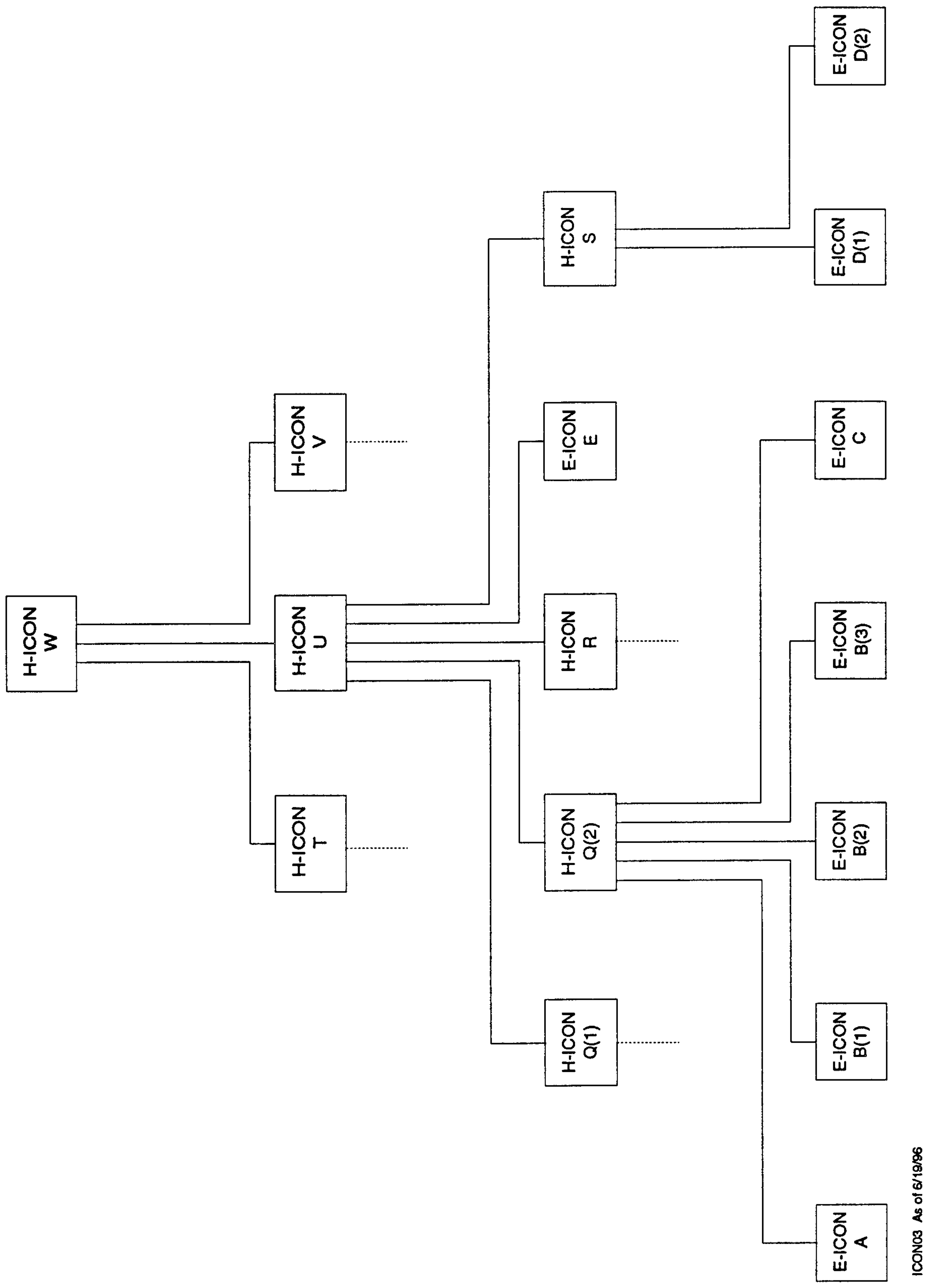

Figure 4-1. Typical hierarchy of a four layer hierarchical icon.

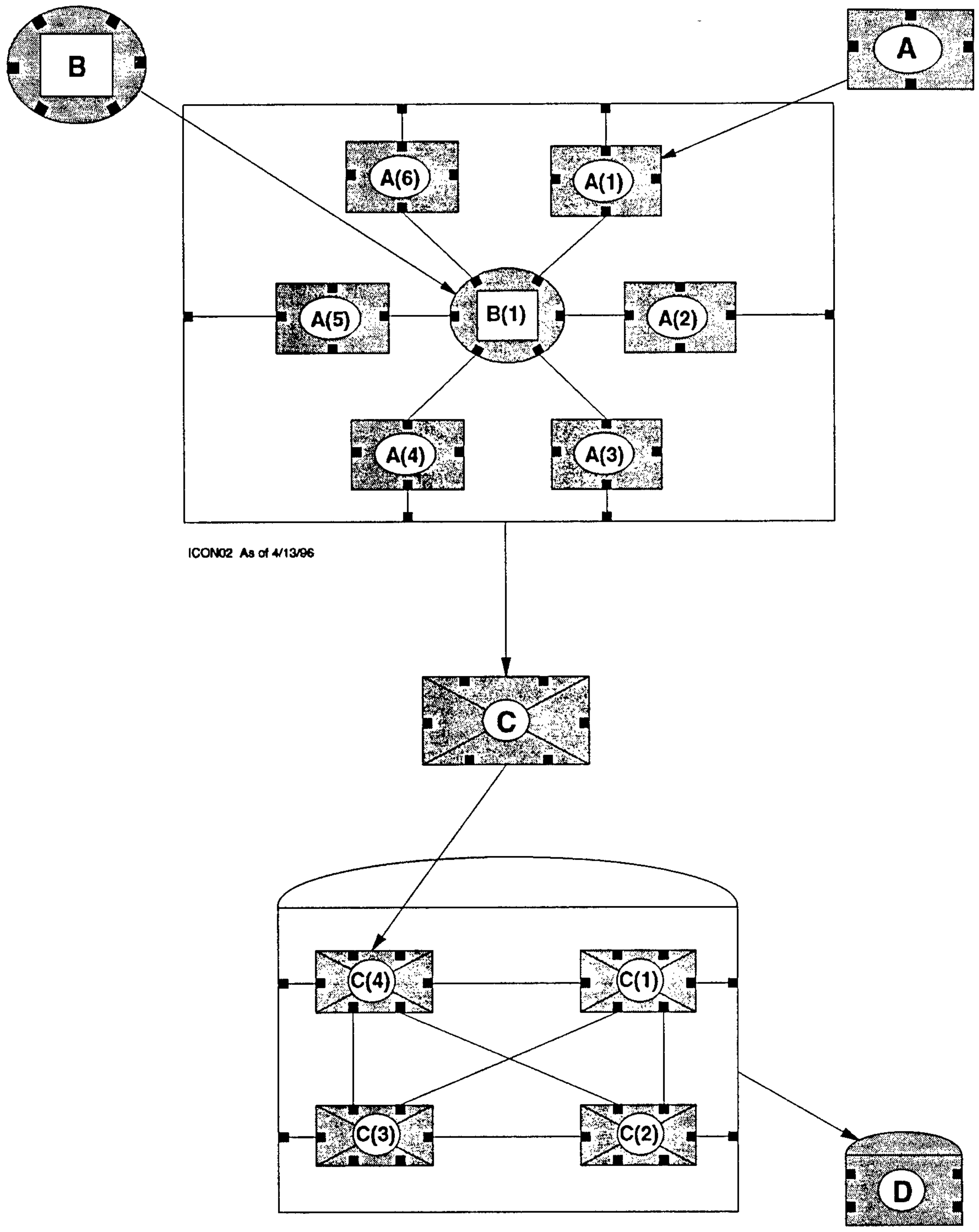

Figure 4-2. Connecting icons to form hierarchical interconnected icons.

ADDITIONAL ICON DESIGN REQUIREMENTS

In the prior section, requirements were defined that provided for the ability to interconnect hierarchical layers of networks where communication system model ports were identified via pin connections on their corresponding icons. With this facility, users will be able to build networks graphically, by interconnecting iconic models of communications equipment, while the simulation is running.

Networks of this type must be fluid in the sense that the network architecture is hierarchical, but dynamically changing. It is hierarchical in the sense that a node can be built up of primitive iconic models being connected together. Such a node can then be represented by a single icon. These nodes can then be interconnected to form larger subnetworks, represented by an icon. At any point in the simulation, an analyst at the workstation may want to detach substructures from the network and attach them to another part of the network.

In addition to the dynamic nature of these hierarchical networks, they are nonhomogeneous. Substructures at any level of the hierarchy can be totally different from their neighbors. In fact, we saw the need for insertion of a given piece of equipment, e.g., a radio, at more than one layer in the hierarchy. In fact, one may want to detach a substructure from one layer of a hierarchy and move it to a lower or higher layer.

Design of the databases required to support dynamic, nonhomogeneous, hierarchical graphic structures is critical to the implementation and success of this project. We must dispense with the notion of hierarchies defined by multiple dimensioned arrays. These hierarchical data structures are not only nonhomogeneous from a data element standpoint, they must also be nonhomogeneous from a quantity standpoint. The number of subordinate elements down any leg of the hierarchy may be totally different from another leg. Some may exist in only a single leg. Trying to accommodate such structures with hierarchies of quantities of elements that, in turn have subordinate quantities of elements, etc., invokes a potentially complex data management problem.

This problem is best solved with flat lists containing the required properties of each type of element along with pointers to their nearest neighbors. Then, these elements can be used to create hierarchies that can be detached and attached by changing their pointers. These elements can be accommodated in a single list that is managed automatically by the graphics database manager for that element. The most complex database management design is for the hierarchical icons. This database design is described in the next section.

This design must also support the interconnection of pinned icons. This involves the requirement that the analyst interconnect icons on a hierarchical basis using pin connections. One must be able to detach a subhierarchy, and attach it somewhere else in the hierarchy. This attachment must involve the interconnection of ports via sequences of pin connections, or links, while the simulation is running. This is covered in the Section 6. The design must also minimize the burden on the modeler who is responsible for building easy-to-use graphical interfaces to complex simulations for use by analysts.

5. HIERARCHICAL ICON SYSTEM DESIGN

Figure 5-1 provides a hierarchy of icons that could be used in RTG. This figure does not show how the icons would appear on the screen. It only indicates what icons will appear. There are four icon *types* in the figure. These are the oval, rectangle, circle, and triangle. In this example, the first three types are *hierarchical;* the triangle is *elementary*. There is a label (e.g., EL-1) and two numbers (e.g., 1 and 24) associated with each icon. The first number is the *instance* number of a particular icon type within a particular subordinate layer of the hierarchy. It is assigned by the modeler. The second number is a unique pointer assigned by the system to every *active* icon.

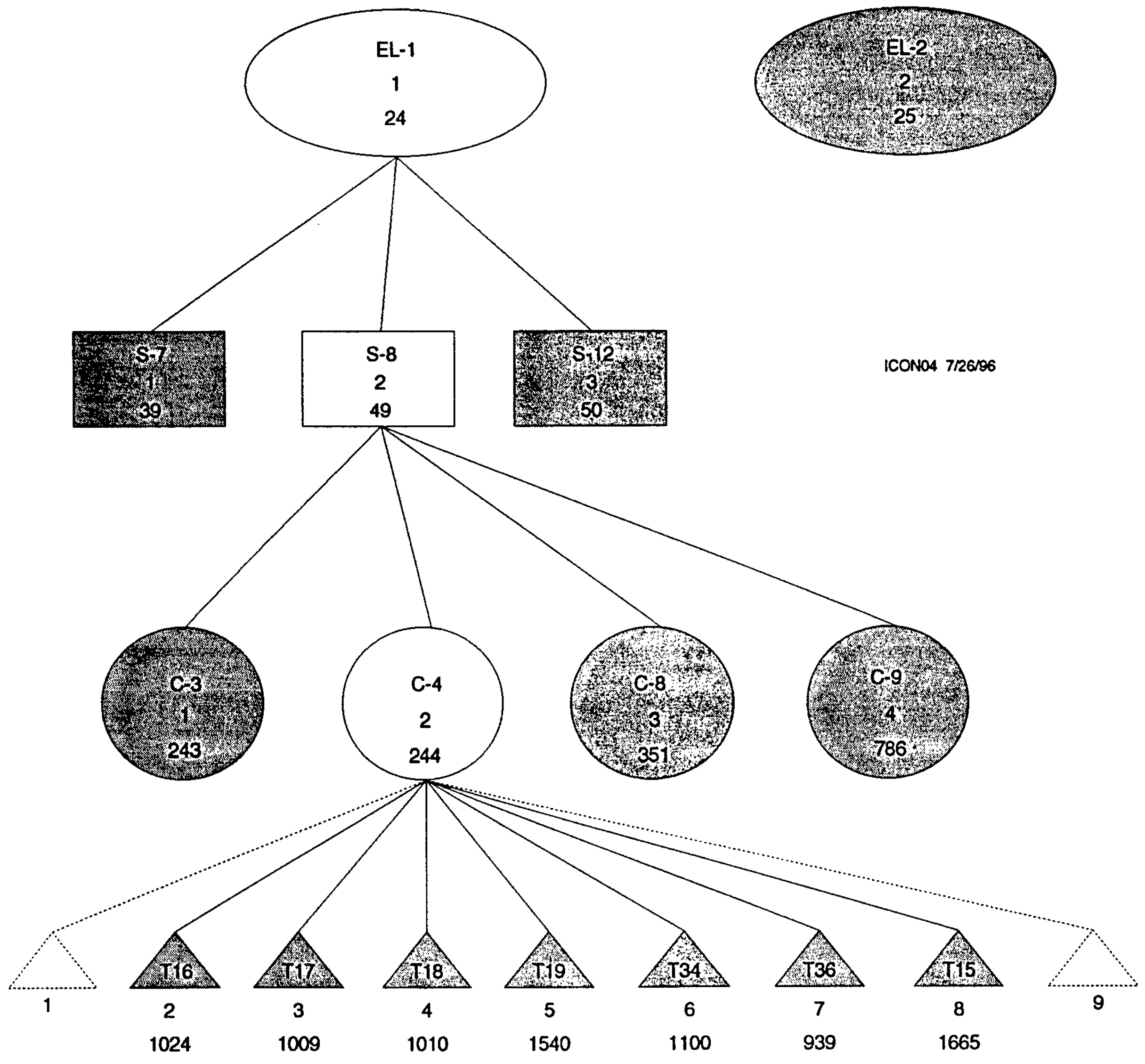

Figure 5-1. Example of an hierarchical icon structure.

ICON INSTANCES AND ICON RECORD POINTERS

When using RTG, icon instances are declared in a resource by the modeler using the attribute type ICON, illustrated as follows.

```
icon_name        QUANTITY(number_of_instances)        ICON
```

Icon_name corresponds to an icon that is either the top level icon in a hierarchy and named in the simulation control specification, or an icon that is contained within the hierarchy. It must reference an icon in the icon library as described in the next section. The number_of_instances of icon_name available to a process is determined by the quantity clause. The ICON type attribute is a four byte integer used as a pointer to an active icon record in the RTG ICON database.

SPECIFYING ICONS IN THE SIMULATION CONTROL SPECIFICATION

Any icons used in a simulation must be identified with a library icon. This is accomplished by declaring those icons in the simulation control specification as follows:

```
ICON icon_name = library_icon_name, size, [, color] [, line_style] [, thickness] [, label(1), ...]
```

If an icon is part of a hierarchy, naming the top level icon brings in all of the subordinate icons. If the user wants to deploy additional instances of an icon beyond those in a hierarchy, the icon must be named here. Also, if the icon_name used in the simulation is different from the icon name in the library, then this statement must be used to relate the icon_name to the library icon. Only those icons named in the simulation control specification are shown in the control window when icons are selected for insertion into the simulation. *Size* can be used to scale the icon from the library to the world space. Color, style and thickness options apply to *homogeneous* icons, i.e., icons whose color, style and thickness properties are the same for elements of an icon, spatially, but can be changed during the course of a simulation. Label areas can also be defined here.

INSERTING AND REMOVING HIERARCHICAL ICONS

When an icon is inserted from the graphics side, a process must be designed by the modeler to handle the event and update the corresponding model databases. This process must store a pointer to the ICON database record in the desired icon instance for use by the modeler using a statement such as:

```
icon_name(instance_pointer) = RTG_ICON_POINTER
```

This pointer is automatically assigned when an icon is inserted from the simulation side. In the case that the icon is a hierarchical icon, the modeler must access the RTG ICON database to get the pointers and any other information needed for subordinate icons. The icon database is described in a later section.

The following statement is used to insert and activate icons from the GSS side.

```
INSERT ICON icon_name(icon_instance) AT x_position, y_position, ...
```

Whenever this statement is invoked, the pointer to that icon's record in the RTG ICON database is assigned to the specified instance of icon_name.

The modeler must declare a sufficient number of instances to support the maximum number of icons anticipated to be active at any instant of time during the simulation. In fact, these instances may be distributed in multiple resources. The INSERT statement simply enters the properties of icon_name taken from the icon library into the RTG ICON database record pointed to by icon_name(icon_instance). The modeler must transfer any properties to be stored elsewhere.

Again, when the icon is hierarchical, the modeler must access the RTG ICON database to get the pointers and any other information needed for subordinate icons. It is up to the modeler to design the databases required to store the corresponding model information. However, one could design records that have the same pointers as those to the RTG ICON database. This eliminates the need for the modeler to manage the linked list pointers on the simulation side. The icon database is described in a later section.

In Figure 5-1, instances 1 and 2 of the oval icon are active. They are active by virtue of having been *inserted* at some point in the past. When hierarchical icons are inserted, all of their subordinate icon layers are also inserted into the RTG ICON database and made active. Any icon at any layer in the hierarchy can be deactivated by *removing* that icon using the REMOVE statement described below. If it is hierarchical, all subordinate layers are removed.

```
REMOVE ICON icon_name(icon_instance)
```

When an icon is removed, it's RTG ICON database record is freed, and the pointer is no longer valid. The record in the database can then be reused to store some other icon instance.

When a hierarchical icon is created using the library management draw facility (off-line), separate records are built for every icon in the hierarchy. Referring to Figure 5-1, there were 9 triangles allocated at the elementary level. Then, at some point during the simulation, triangles 1 and 9 were removed and no longer have RTG records active in the simulation. Any of the 7 remaining triangles can be deactivated by a remove. Conversely, we can add triangles back into the hierarchy by first inserting them, and then *attaching* them to the hierarchy. This process is described in the sections that follow.

COVERING AND UNCOVERING HIERARCHICAL ICONS

In the case of a hierarchy, subordinate icons may be made *visible* in place of their active superiors by *uncovering* the superior using the UNCOVER statement:

UNCOVER ICON icon_name(icon_instance) .

When the cover was lifted from oval 1, the next layer was made visible and the oval disappeared from view. Similarly with rectangle 2 and circle 2. Although these icons are not visible, they are still active. Only the shaded icons appear on the screen by virtue of having the covers lifted on their superiors up the line. If there were no hierarchical icons used in the simulation, then all active icons would be visible on the screen. The UNCOVER statement only applies to active visible superior icons having a subordinate hierarchy.

Conversely, subordinate icons may be covered by their superior using the COVER statement:

COVER ICON icon_name(icon_instance)

where icon_name refers to the superior icon that is to be used to cover its subordinates. The COVER statement only applies to active superior icons having visible subordinates.

MODIFYING HIERARCHICAL ICONS

Hierarchical icons may be updated, reshaped, etc. using the corresponding RTG statements just as elementary icons not part of a hierarchy. When these operations are applied, they are applied to the layer specified. As an example, consider the UPDATE statement.

UPDATE ICON icon_name(icon_instance) TO x_position, y_position, ...

If icon_name(icon_instance) is a hierarchical icon, it is updated just like any other icon. If it is visible, none of the subordinate icons are updated with this statement. If it is not visible, all of the subordinate icons down to the visible layer (inclusive) are updated along with this icon. Those below the visible layer are not.

HIERARCHICAL ICON DATABASE DESIGN

From a user standpoint, every icon may be a unique independent entity with a distinguishable identity. Even though every triangle icon along the bottom of Figure 5-1 may have the same shape and color, it may have different properties from a modeling standpoint, and be identified differently with different labels. If the triangles represented tanks in a military operation, tanks T34 and T36 may have been moved from tank platoon P9 to tank platoon P7.

The individual tanks will maintain the same properties they had when they were moved, e.g., a given amount of fuel and ammunition.

To support these kinds of operations, one must be able to move icons from one structure to another. Thus the data structure must allow a fluid flow of particular icon types among similar structures in a hierarchy. The database design in Figure 5-2 supports this operation. All active icons are kept in this database, from the time they are inserted until the time they are removed.

ATTACHING ICONS TO - AND DETACHING ICONS FROM - HIERARCHIES

As indicated in the above example, moving individual tanks from one platoon to another requires the ability to detach icons from one hierarchy and attach them to another. This is accomplished using the DETACH and ATTACH statements defined below.

DETACH icon_s(instance_s)

This statement automatically removes subordinate icon_s(instance_s) from its position in the hierarchy with its immediate superior. When appropriate, its relative properties are automatically replaced by their corresponding actual properties in its database record. At this point, it becomes a free icon - not part of any superior hierarchy. Note that the detached icon may itself be a superior icon with a hierarchy that remains with it. Icon_s can now be manipulated independent of its former superior hierarchy.

When an icon is detached from a hierarchy, it becomes visible and superior. If it is subsequently desired to attach this icon into the same or a different hierarchy, then one uses the ATTACH statement.

ATTACH icon_s(instance_s) TO icon_p(instance_p)

In this case, icon_s(instance_s) is attached as an immediate subordinate in a hierarchy to superior icon_p(instance_p). If icon_s(instance_s) is a hierarchical icon, then its hierarchy is attached to icon_p(instance_p) accordingly.

When an icon is attached to a hierarchy, it becomes a subordinate, and its visibility will depend upon the visibility of the layer to which it was attached. Its actual properties may be replaced by the appropriate relative properties in its database record if it is no longer visible.

The ATTACH and DETACH statements automatically update the hierarchy properties in the database records (Figure 5-2) for all of the appropriate icons.

```
RESOURCE NAME: ACTIVE_ICON_DATABASE

COMMENT:  CONTAINS A DATABASE RECORD FOR
          EACH ACTIVE ICON IN THE SIMULATION

ICON_STATE_CONTROLS
    1   MAXIMUM_ICONS                  INDEX *** MAXIMUM - 2000
    1   INITIAL_ICONS                  INDEX *** INITIAL ICON DEFINITIONS
    1   ACTIVE_ICONS                   INDEX *** CURRENT NUMBER ACTIVE
    1   VISIBLE_ICONS                  INDEX *** CURRENT NUMBER VISIBLE
    1   ICON_LAYERS                    INDEX *** NUMBER OF ICON LAYERS
    1   RTG_ICON_POINTER               INDEX *** WORKING ICON POINTER
    1   PIN_VIEW                       STATUS SHOWING
                                              HIDING
    1   PIN_LABELS                     STATUS SHOWING
                                              HIDING

ICON_STATE   QUANTITY(2000) *** POINTER ALSO USED AS DRAW LIST NUMBER
    1   ICON_LIBRARY_NAME              CHAR 24 *** LIBRARY NAME
    1   ICON_SIMULATION_NAME           CHAR 24 *** NAME ASSIGNED IN SIMULATION
    1   ICON_TYPE                      CHAR 1
                 ALIAS  HOMOGENEOUS          VALUE 'H'
                 ALIAS  NONHOMOGENEOUS       VALUE 'N'
    1   ICON_CONNECTIONS               CHAR 1
                 ALIAS  PINNED               VALUE 'P'
                 ALIAS  UNPINNED             VALUE 'U'
    1   ICON_STATE                     CHAR 1
                 ALIAS  VISIBLE              VALUE 'V'
                 ALIAS  INVISIBLE            VALUE 'I'
    1   ICON_COORDINATES               CHAR 1
                 ALIAS  ACTUAL               VALUE 'A'
                 ALIAS  RELATIVE             VALUE 'R'
                 ALIAS  BOTH                 VALUE 'B'
    1   ICON_ACTUAL_POSITION
        2   ACTUAL_X                   INTEGER
        2   ACTUAL_Y                   INTEGER
        2   ACTUAL_Z                   INTEGER
    1   ICON_RELATIVE_POSITION
        2   RELATIVE_X                 INTEGER
        2   RELATIVE_Y                 INTEGER
        2   RELATIVE_Z                 INTEGER
    1   ICON_SCALING
        2   ICON_SCALE_X               REAL
        2   ICON_SCALE_Y               REAL
        2   ICON_SCALE_Z               REAL
    1   ICON_ROTATION
        2   ICON_ROTATE_X              INTEGER
        2   ICON_ROTATE_Y              INTEGER
        2   ICON_ROTATE_Z              INTEGER
    1   ICON_LABEL   QUANTITY(3)
        2   LABEL_NAME                 CHAR 12
        2   LABEL_POSITION                  *** RELATIVE TO ICON ORIGIN
            3   LABEL_X                INTEGER
            3   LABEL_Y                INTEGER
            3   LABEL_Z                INTEGER
    1   HOMOGENEOUS_PROPERTIES
        2   ICON_COLOR                 INDEX_1
        2   ICON_STYLE                 INDEX_1
        2   ICON_THICKNESS             INDEX_1
        2   ICON_BLINK_PRIORITY        INDEX_1
```

Figure 5-2A. Hierarchical icon database structure.

```
1    PIN_CONNECTIONS
     2    NUMBER_OF_PINS              INDEX_1
     2    FIRST_PIN_POINTER           INDEX_1   *** POINTER TO PIN LIST
1    HIERARCHY_PROPERTIES
     2    SUPERIOR_LAYER
          3  HIERARCHY                CHAR 1
                ALIAS EXISTS               VALUE 'E'
                ALIAS NON_EXISTANT         VALUE 'N'
          3  SUPERIOR_ICON            INDEX    *** POINTS BACK TO ICON
                                               *** IN THE PRIOR LAYER
     2    CURRENT_LAYER
          3  PRIOR_ICON               INDEX    *** POINTS TO PRIOR ICON
                ALIAS  NON_EXISTANT        VALUE 0
          3  NEXT_ICON                INDEX    *** POINTS TO NEXT ICON
                ALIAS  NON_EXISTANT        VALUE 0
     2    SUBORDINATE_LAYER
          3  HIERARCHY                CHAR 1
                ALIAS EXISTS               VALUE 'E'
                ALIAS NON_EXISTANT         VALUE 'N'
          3  FIRST_SUBORDINATE_ICON INDEX      *** POINTS TO FIRST ICON
                                               *** IN THE NEXT LAYER
                ALIAS  NON_EXISTANT        VALUE 0
```

Figure 5-2B. Hierarchical icon database structure.

UPDATING RELATIVE AND ACTUAL PROPERTIES OF HIERARCHICAL ICONS

Certain properties of subordinate icons are relative to their superiors in a hierarchy. These are the position, scaling and rotation states. If a superior icon is translated, scaled or rotated, all of the subordinate icons are also assumed to be affected accordingly. Using GSS, the modeler is expected to update these properties in the corresponding model databases as he deems necessary.

In RTG, subordinate icons that are not visible store these properties relative to their superior at the next level up. When their superiors are updated, they remain untouched. When their immediate superior is uncovered, then the properties of that superior are effectively added to the subordinates before they are made visible so that they appear with their actual properties. Conversely, when visible subordinates are covered by a superior icon, the properties of the superior are effectively subtracted from the subordinates so the subordinates store those properties relative to the superior following the cover operation.

In summary, the visible icons and their superiors always have the actual values of those properties that are otherwise stored as relative when they are covered by superiors.

INTERACTIVE SIMULATION SYNCHRONIZATION

When allowing users to interactive with the simulation, one must be concerned with synchronization of actions so that the user does not do something to invalidate the simultaneous action of the simulation, and vice versa. This can be achieved by giving both the modeler and the interactive user the ability to deactivate or "freeze" the other side so that such simultaneous actions do not occur.

On the interactive RTG side, this is accomplished by the user selecting the simulation control option to *pause* the simulation while actions are taken through the interactive graphics facility. When the user has completed the desired actions, the simulation can be continued simply by selecting the *continue* the simulation.

In the simulation, the modeler can pause the graphics using the PAUSE GRAPHICS statement. When this statement is encountered, a message is produced on the RTG display indicating that interactive inputs have been inhibited by the simulation, at which time RTG is suspended. When the modeler issues the statement RESUME GRAPHICS, RTG is resumed, the message is removed from the display, and the user can proceed with interactive inputs.

FUTURE CONSIDERATIONS

The COVER and UNCOVER statements can be appended with a [WITH OUTLINE] clause. This would cause an outline of the superior icon to appear when the subordinates are visible. This would be most helpful when working with pinned icons.

The icon INSERT, UPDATE, and other property change statements may be appended with

```
[ UPDATING ACTUAL VALUES ]
```

to indicate that the actual coordinates must be updated for covered icons. Otherwise, the interpretation of these quantities will depend upon the visability of icons in the hierarchy.

HIERARCHICAL ICON DESIGN SUMMARY

As we have described in Section 4, design of the databases required to support dynamic, nonhomogenous, hierarchical graphic structures is critical to the implementation and success of this project. Based upon design requirements for using icons, we have dispensed with the notion of hierarchies defined by multiple dimensioned arrays as in programming languages. The required hierarchical data structures are not only nonhomogenous from a data element standpoint, they must also be nonhomogenous from a quantity standpoint. The number of subordinate elements down any leg of an iconic model hierarchy may be totally different from another leg. Some may exist in only a single leg. Although the total quantity of any particular element may be fixed, the quantity of elements in any leg may vary with time as subhierarchies and elements are detached and reattached elsewhere.

To meet these requirements, the architecture presented here, and the corresponding database design, minimizes the burden on the modeler who is responsible for building easy-to-use graphical interfaces to complex simulations for use by analysts. It provides an unprecedented level of flexibility for the analyst who can move any element of an iconic model hierarchy from one spot to another in the hierarchy. This facility will help to revolutionize the use of modeling and simulation in the analysis, design, and test of communication network architectures.

6. IMPLEMENTING PIN CONNECTIONS

As shown in Figure 5-2A, the icon record contains the number of pins associated with that icon, and a pointer to the first pin in the list of pins associated with that icon. The data structure containing all of the pin lists for all objects in the simulation is shown in Figure 6-1. This database contains the information pertinent to each pin in each pin list. It contains the pin coordinates relative to the object (icon, line, or instrument), a pointer back to that object, and a pointer to the next pin in the list, if one exists.

A pin on one elementary icon can be connected directly to a pin on another elementary icon. In this case, the link is specified by these two pins. In the case that a hierarchical icon overlays the elementary icon, then interconnections outside the hierarchical icon must end on pins on that hierarchical icon, reference Figure 6-2.

The ability to show or hide the pins will be implemented on an overall basis to start. Choice by object type or individual object will be considered later.

```
RESOURCE NAME: PIN_DATABASE

COMMENT:  CONTAINS THE PIN LISTS FOR EACH
          PINNED ICON, LINE, AND INSTRUMENT.
          USES SCALING AND ROTATION OF THE OBJECT


PIN_DATA      QUANTITY(5000)
    1   PIN_POSITION  *** RELATIVE TO OBJECT COORDINATES
        2   PIN_X               INTEGER
        2   PIN_Y               INTEGER
        2   PIN_Z               INTEGER
    1   PIN_ID
        2   PIN_NUMBER          INDEX    *** USER NUMBER
        2   PIN_LABEL_POS       STATUS TOP, BOTTOM
                                       LEFT, RIGHT
    1   PINNED_OBJECT           CHAR 1
             ALIAS  ICON               VALUE 'I'
             ALIAS  LINE               VALUE 'L'
             ALIAS  INSTRUMENT         VALUE 'N'
    1   OBJECT_POINTER          INDEX  *** POINTER TO OBJECT DATABASE RECORD
    1   NEXT_PIN_POINTER        INDEX  *** POINTER TO NEXT PIN IN LIST
             ALIAS  NON_EXISTANT       VALUE '0'
```

Figure 6-1. Pin table data structure.

```
RESOURCE NAME: LINK_DATABASE

COMMENT:  CONTAINS THE LINK LISTS FOR EACH
          PINNED INTERCONNECTION


LINK_DATA     QUANTITY(4000)
    1   FIRST_PIN                INDEX   *** POINTER TO FIRST PIN
    1   SECOND_PIN               INDEX   *** POINTER TO SECOND PIN
    1   LINE_POINTER             INDEX   *** POINTER TO LINE CONNECTING PINS
    1   ADJACENT_LINK_POINTER    INDEX   *** POINTER TO NEXT ADJACENT LINK
                                         ***                OFF FIRST PIN
             ALIAS  NON_EXISTANT     VALUE '0'
    1   NEXT_LINK_POINTER        INDEX   *** POINTER TO FIRST LINK
                                         ***           OFF SECOND PIN
             ALIAS  NON_EXISTANT     VALUE '0'
```

Figure 6-2. Link table data structure.

In the drawing board facility, users will be able to rubber-band lines, e.g., occur when connection lines are horizontal or vertical, implying right angle bends in lines. Pins will be used for connecting sequences of lines.

7. INTERACTIVE MENU SELECTION

Most of the selection process associated with performing a function can be implemented using hierarchical menus. Figure 7-1 serves to illustrate this process.

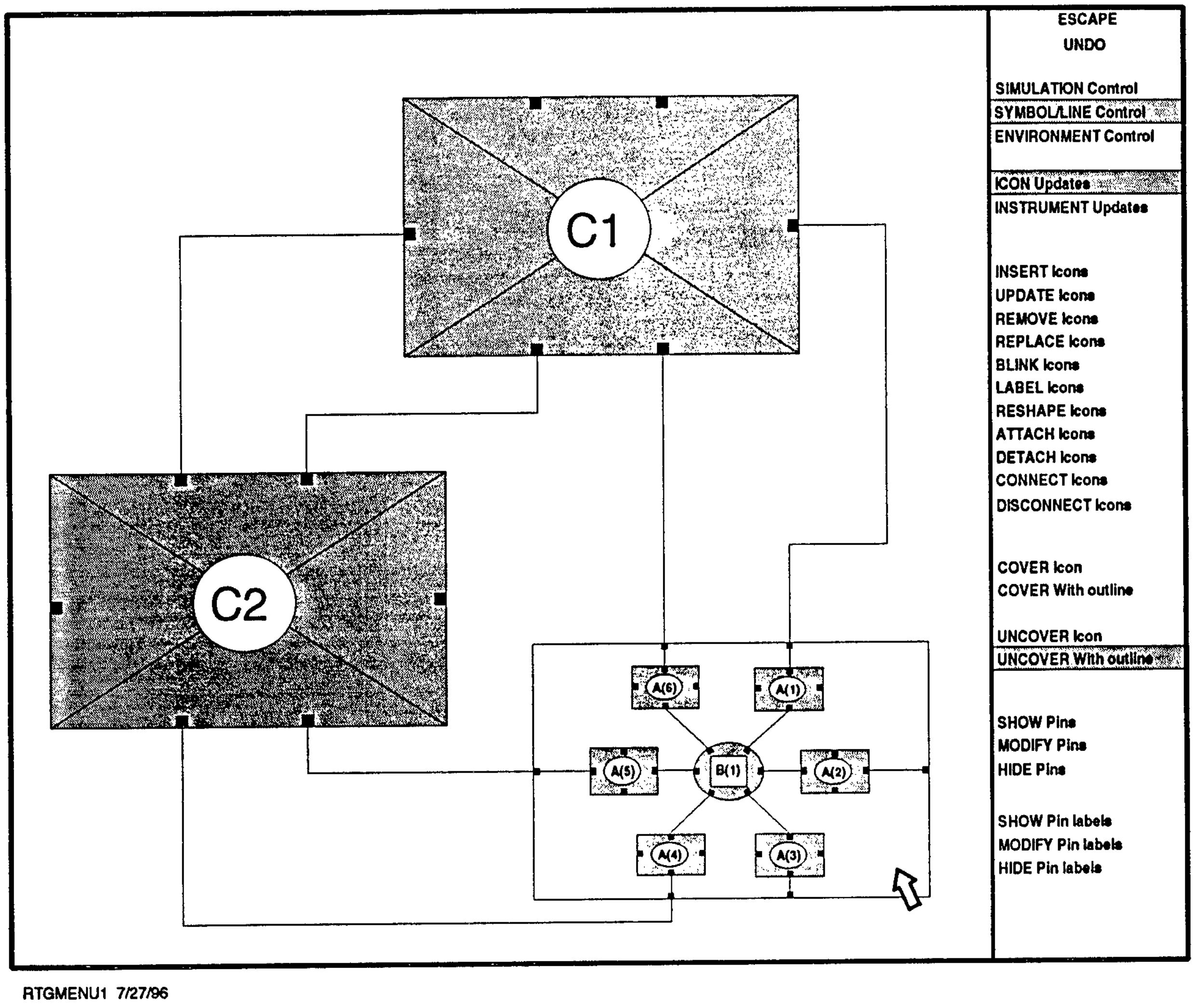

Figure 7-1. Example of an hierarchical menu selection procedure.

A menu is shown to the right of the graphics display in Figure 7-1. This is the way the user is prompted using RTG. With a relatively narrow window to the right, many prompts can be exercised and many icons can be put up for selection by the user without taking away too much of the graphical picture to the left. We propose to stay with this proven design philosophy.

As the user makes selections from the hierarchy of menus, a state table is updated that determines the state of the system at that time. For example, looking at Figure 7-1, the user has selected SYMBOL/LINE Control from the top level menu. ICON Updates was then selected from the next level menu. UNCOVER With outline was selected from the next level. Note that in levels, some entries can be set apart from others. This does not remove them from that level. This functional partitioning is for the convenience of understanding and selection by the user. The implementation of levels behind the scenes is for the convenience of the software architecture, allowing for easy upgrade and support.

At each level in the hierarchy, a state is stored. When the bottom of the selection path is reached, a graphic function is performed. This graphic function may require a separate elementary module, or be part of another elementary module. Most of these graphic functions will be implemented using one or more utility modules described in Section 9.

The existing hierarchical menus in GSS are table driven using X-Windows and are easy to modify. An example of the hierarchical menu table that can define the entries for RTG is shown in Figure 7-2. The left most column is the entry number. The second column indicates the number of entries in a single menu. The third column holds the keypress button ID. The fourth column contains the label in the menu. The last column holds a pointer to the top of the next menu in the hierarchy. If this column contains a zero, the bottom of the hierarchy has been reached.

0001	03	S	SIMULATION Control	0004
0002	00	Y	SYMBOL/LINE Control	0012
0003	00	E	ENVIRONMENT Control	0053
0004	06	U	USER DEFINED MENUS	0000
0005	00	D	DEFINE Simulation	0010
0006	00	R	RESTART Simulation	0000
0007	00	H	HALT/CONTINUE Simulation	0000
0008	00	S	STOP/CLOSE Simulation	0000
0009	00	T	TERMINATE Simulation	0000
0010	02	W	WRITE RECORDING FILE	0000
0011	00	R	READ REPLAY FILE	0000
0012	02	I	ICON Udates	0019
0013	00	N	INSTRUMENT Udates	0039
0014	25	I	INSERT Icon(s)	0000
0015	00	U	UPDATE Icon(s)	0000
0016	00	R	REMOVE Icon(s)	0000
0017	00	P	REPLACE Icon(s)	0000
0018	00	B	BLINK Icon(s)	0000
0019	00	L	LABEL Icon(s)	0000
0020	00	T	RESHAPE Icon(s)	0000
0021	00	K	ATTACH Icon(s)	0000
0022	00	S	DETACH Icon(s)	0000
0023	00	C	CONNECT Icon(s)	0049
0024	00	D	DISCONNECT Icon(s)	0050
0025	00			0099
0026	00	sC	COVER Icon	0000
0027	00	cC	COVER Icon with outline	0000
0028	00			0099
0029	00	sU	UNCOVER Icon	0000
0030	00	cU	UNCOVER Icon with outline	0000
0031	00			0099
0032	00	sS	SHOW Pins	0000
0033	00	sM	MODIFY Pins	0000
0034	00	sH	HIDE Pins	0000
0035	00			0099
0036	00	cS	SHOW Pin labels	0000
0037	00	cM	MODIFY Pin labels	0000
0038	00	cH	HIDE Pin labels	0000

MENUTAB3 6/17/96

Figure 7-2. Example of a VSE hierarchical menu table.

PIN SELECTION AND INTERCONNECTION

Figure 7-1 shows a network connected with pins. The pins are inserted as part of the icon when the icon is created or updated in the drawing board. It is up to the user creating the icon to determine where the pins are placed on the icon. Not shown are the labels that are implicit with each pin in an icon.

In the example network shown in the figure, a pin is associated with a port, where each port is a separately identifiable entity in the model of a piece of communication equipment. Each pin associated with an instance of an icon must be separately identifiable. This implies that the user must have the ability to associate a label with each pin.

Because of the potential clutter of numerous pins and labels, a facility will be incorporated to hide pins as well as labels on an all-on or all-off basis. In Figure 7-1, the pins are shown but the labels are hidden. The labels will only be shown when the pins are shown.

To connect pins, the user must be able to *rubber-band* the connect lines. This will be accomplished by allowing the user to pick a line and pull that line at the point where it has been picked so as to produce lines aligned with the horizontal and vertical axis and connected at right angles. One may start with a straight line that lies on an angle between pins, and rubber band that line until it is a sequence of horizontal and vertical connect lines as shown in Figure 7-1.

PICKING AND SELECTING ICONS, LINES, AND INSTRUMENTS

Over the past decade, various approaches have evolved for picking and selecting graphic objects on the display screen. We are concerned about picking and selecting icons, lines, and instruments. We are especially interested in the selection of objects for grouping into hierarchies, and adding and deleting objects from these groups. To accomplish this, the functions that users will typically want to perform can be categorized as follows:

- Select a particular object.
- Select a group of objects by drawing a box around them using the mouse.
- Add an object to the current selection.
- Add a group of objects to the current selection.
- Delete an object from a selection.
- Pick a single object from a group of overlapping objects.

An approach has evolved that has become virtually a de facto standard for implementing these functions. The details of this approach are described below.

1. All selection is done by pointing with the mouse cursor and using the left mouse button. In what follows, cursor means the *cursor* tied to the mouse, and *button* means the left mouse button.

2. Clicking on an object selects it and deselects all other currently selected objects. If the cursor is on top of multiple objects, the smallest is selected.

3. Clicking down where there is no object, holding the button down while dragging the cursor, and then releasing the button selects all the objects in a screen-aligned rectangle whose corners are determined by the cursor positions when the button went down and where it came up. This is called a *box selection*. Only those objects completely inside the boxed-out region are selected.

4. If the Shift key is held down and the user clicks on an object that is not currently selected, that object is added to the selected list. If the clicked-upon object was already selected, it is deleted from the selection list.
5. If a box selection is performed with the Shift key pressed, the objects boxed out are added to the current selection.
6. If the Shift key is held during this box selection, the objects enclosed in the box region are added to the current selection.
7. Finally, if the user clicks on multiple objects, select just one of them. If the cursor isn't moved (or not moved more than a pixel), and the user clicks again in the same place, deselect the object originally selected, and select a different object under the cursor (the next in order by size). Use repeated clicks at the same point to cycle through all the possibilities.

Since the approach described above is common to most mouse oriented graphics software today, and because it has evolved through heavy utilization by millions of users over many years, we will adhere to it where we can. In some cases, we will require special procedures for picking and selection.

8. SYSTEM ARCHITECTURAL DESIGN

In this section we describe the overall system architectural design. Since the icons used in the run-time system must first be created and entered into a library, the architecture contains an icon library management system as well as the run-time system. We will first describe the library management system that is used to create hierarchies of icons as well as the icons themselves. This will be followed by a description of the architecture of the run-time system itself.

ICON LIBRARY MANAGEMENT FACILITY

Creation and modification of hierarchical icons is best accomplished using an icon draw utility as described in the first R & D Status Report for this project. In that report, we referred to this as the icon *drawing board,* a utility within the Icon Library Management Facility. The top level architecture of this facility is shown in Figure 8-1. When icons are created using the drawing board, a database is built containing the information required to draw the icons during simulation run-time. Also created are a set of graphics language (GL) source statements comprising a *draw list* for each icon.

At simulation run-time, the GL source statements reference an icon database that controls the transformations required to insert, update, and reshape icons. Draw lists are embedded in a software module that is generated automatically by the Icon Library Manager. When compiled, a draw list produces special run-time draw code that can be called for rapid draw of an icon.

The GL draw list source for each icon is stored in the DRAW LIST DATABASE as shown in Figure 8-1. The draw list defines the icon properties, such as line color, style and thickness, and the vertices for the icon draws. These draw lists are incorporated into a simulation when those icons are referenced in the simulation's control specification. For each icon specified, its corresponding draw list module is dynamically included into the RTG module for drawing that icon at run-time.

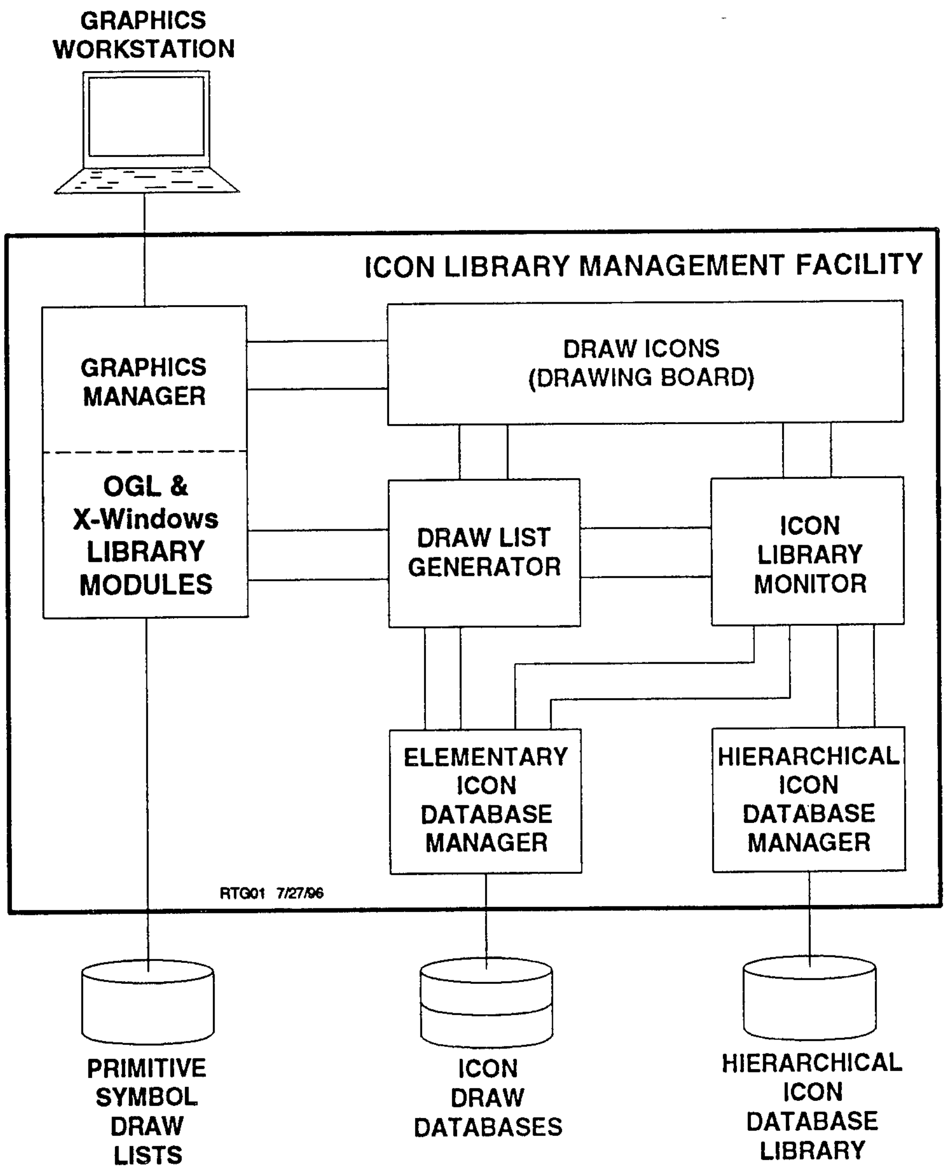
GRAPHICS WORKSTATION
ICON LIBRARY MANAGEMENT FACILITY
GRAPHICS MANAGER
OGL & X-Windows LIBRARY MODULES
DRAW ICONS (DRAWING BOARD)
DRAW LIST GENERATOR
ICON LIBRARY MONITOR
ELEMENTARY ICON DATABASE MANAGER
HIERARCHICAL ICON DATABASE MANAGER
RTG01 7/27/96
PRIMITIVE SYMBOL DRAW LISTS
ICON DRAW DATABASES
HIERARCHICAL ICON DATABASE LIBRARY

Figure 8-1. Icon Library Management Facility - top level architecture.

TOP-LEVEL RUN-TIME GRAPHICS ARCHITECTURE

Specifying Coordinate Transformation Ratios

There are certain design areas that must be resolved at the top level architecture. These include transformations between Basic Draw Space (BDS), Normal View Space (NVS) and Relative View Space (RVS) as described in the last status report. These transformations will affect the input and output of mouse positions, as well as changing the relative size of items in the window when panning, zooming, or reshaping the window. The ratio of transformations from BDS to NVS must be specified by the user for each icon declared in the simulation control specification. The user must also specify the ranges of x, y, and z coordinates for the entire simulation, i.e., the *world view*. As an option, the user can specify the *initial viewing window* by specifying the lower left-hand corner and upper right-hand corner in terms of the x, y, and z world view coordinates. The default is the entire range, i.e., the world view fully zoomed out.

Simulation - Graphics Interface

The current RTG facility uses a "communication channel" interface between the simulation and RTG. This is because the original implementation used the DEC VAX to house the simulation and the Silicon Graphics workstation to run the graphics. We are now building networked simulations using multiple SGI workstations that each house a simulation, and can also run in client-server mode. Neither of these cases justify the communication channel design between GSS and RTG, a design that requires duplication of the graphics data base on both sides of the channel.

The proposed design for RTG provides for a single shared database, using a pair of shared resources as shown in Figure 8-2. With this facility, that part of the graphic database required by the modeler can be updated directly, eliminating the need for the SHOW and COLLECT statements to update the graphics database on the simulation side.

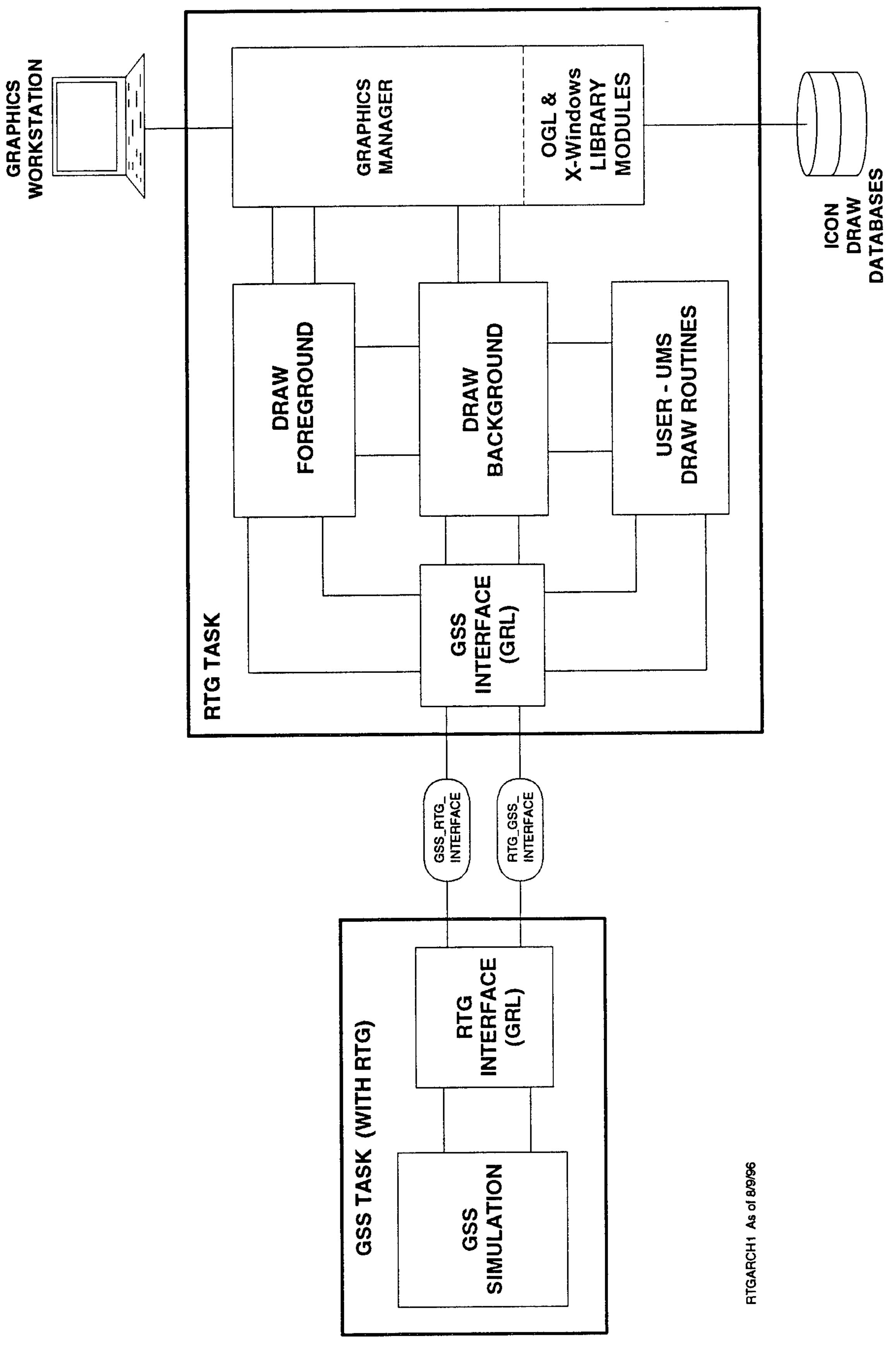

Figure 8-2. GSS-RTG Run-Time System Architecture.

Foreground Versus Background Draws

When an icon is referenced in the simulation control specification, its GL draw list source code and corresponding database will be incorporated into the DRAW_FOREGROUND module in the simulation as shown in Figure 8-2. The draw function can be called upon when redrawing the foreground, or for individual calls to insert or update an icon when a redraw of the entire foreground is not required. Foreground draw speed will be enhanced by this dynamic approach to generating icon draws.

Drawing an icon that was not active, and therefore not drawn before, can be accomplished by an immediate draw of that icon alone without redrawing the rest of the foreground. If an icon changes its position, or is rotated, scaled, or deleted, then the total foreground must be redrawn. When these draw events are called for, the icon database is updated and the decision is made whether to draw a single icon, or to redraw the foreground.

Run-Time Draw State Changes

At simulation run-time, icon draws can be initiated interactively by the user at the work station, or by process statements in a GSS model. In either case, these graphic events first cause the icon database to be updated. The following database elements can be updated for either type of icon, i.e., a color-filled icon or a wire-frame icon.

- Draw Type (color-filled versus wire-frame)
- Connection Type (pinned versus unpinned)
- Position (X, Y, Z)
- Scaling (Sx, Sy, Sz)
- Rotation(Θx, Θy, Θz)
- Blink Priority (on, off)
- Labels (text box)

For wire-frame icons, i.e., those that are not color-filled but have all lines the same color, the additional database items can be updated.

- Color (allowed color index numbers)
- Line Style (allowed stipple numbers)
- Line Thickness (allowed thickness numbers)

Updating The Icon State Database

The icon manipulation statements are categorized below. These are essentially the same as those in RTG with the removal of COLLECT and SHOW, and the addition of ATTACH, DETACH, CONNECT, DISCONNECT, COVER, and UNCOVER. As described above, COLLECT and SHOW will not be needed if the required elements of the graphics database are stored in the shared resources between the simulation and RTG.

STATEMENT	ACTION	DRAW
INSERT	Inserts an icon at specified position	Direct
UPDATE	Updates certain icon properties	depends
REMOVE	Removes an icon	Redraw
REPLACE	Replaces an icon with a new icon type	Redraw
BLINK	Causes an icon to blink or stop blinking	Direct
LABEL	Changes an icon text label	Direct
RESHAPE	Updates certain icon properties	depends
ATTACH	Attaches an icon to a hierarchy	depends
DETACH	Detaches an icon from a hierarchy	depends
CONNECT	Connects icons with lines	depends
DISCONNECT	Disconnects lines between icons	depends
COVER	Covers an icon layer with another icon	Redraw
UNCOVER	Uncovers an icon layer	Redraw

ICON DRAW CONSIDERATIONS

Icons are drawn using the DRAW_FOREGROUND module of Figure 8-2. A more detailed representation of this module is shown in Figure 8-3 to illustrate its use. Databases for icons are built dynamically to support the simulation control specification.

When a process invokes a change to an icon, the result will be an update of the icon database, followed by a draw of that icon, or a redraw of the foreground. If the icon is moved, it is much easier and likely faster to redraw the foreground. Otherwise, one has to delete the icon, replace all former elements that the icon overlayed, and then redraw the icon in its new spot. Replacing the former background is a very difficult problem, particularly if there are many draws intersecting that icon and there are many foreground items to track. Graphics experts recommend redrawing the foreground.

Complex foreground redraws can be designed to take advantage of hardware draw facilities when they are available, e.g., the Silicon Graphics workstation. When such facilities are available, the foreground draws can be very fast. Using Open GL, draw lists are automatically processed to take advantage of whatever hardware is available. In addition, Open GL provides for hierarchical draw lists so that complex hierarchical icons are supported directly.

The examples that follow indicate how one can partition the design so as to minimize and isolate the Open GL graphics calls within C routines.

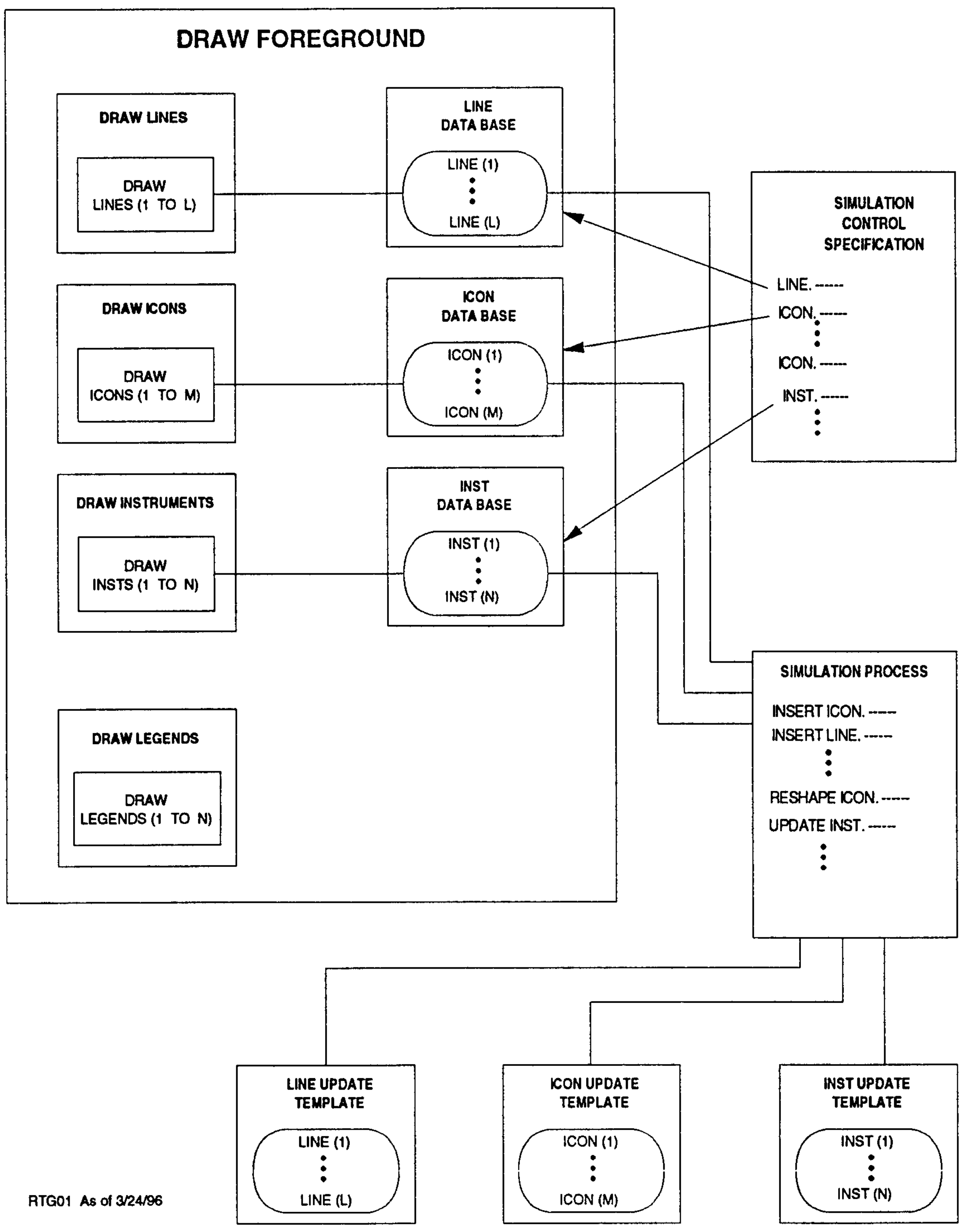

Figure 8-3. Hierarchical grouping of icons.

Example of calls from VSE to ICON draw routines using Open GL.

```
***   VSE PROCESS TO DRAW ICONS   ***

            .
            .
            .

DRAW_ALL_ICONS     ***  ALL INSTANCES OF EVERY ICON
    EXECUTE DRAW_ALL_ICON_INSTANCES
        INCREMENTING ICON_ID_POINTER
            TO MAX_ICON_TYPES_IN_USE

DRAW_ALL_ICON_INSTANCES   ***  OF THE SELECTED ICON
    ICON_NUMBER = ICON_ID(ICON_ID_POINTER)
    MAXIMUM_INSTANCES = MAX_ICON_INSTANCES(ICON_ID_POINTER)
    EXECUTE DRAW_ICON_INSTANCE
        INCREMENTING ICON_INSTANCE
            TO MAXIMUM_INSTANCES

DRAW_ICON_INSTANCE
    CALL DRAW_ICON_INSTANCE USING ICON_DRAW_DATABASE
```

```
***   C ROUTINE TO DRAW AN INSTANCE OF AN ICON   ***

# include ICON_DRAW_DATABASE

void DRAW_ICON_INSTANCE();
{

      /* SET THE DYNAMIC ICON PROPERTIES */

            glLineWidth(LINE_WIDTH);

            glLineStipple(1x3f07);
            glEnable(GL_LINE_STIPPLE);

            glColor3f(COLOR_R, COLOR_G, COLOR_B);

            /* PERFORM THE TRANSFORMATIONS IN REVERSE ORDER */

            glTranslatefv(TRANSLATE_VECTOR);

            glRotatefv(ROTATE_VECTOR);

            glScalefv(SCALE_VECTOR);


      /* CALL THE SPECIFIC ICON DRAW ROUTINE */

      glCallList(ICON_NUMBER);
}
```

An example of an Open GL routine to build a CallList for drawing a particular icon type. There will be one routine for every icon in the icon library. These lists can be drawn hierarchically, i.e., a CallList can contain other CallLists. These routines must be compiled prior to their use. They are then fixed. Properties of the icons can be changed dynamically, as shown in the prior section.

```
# include ICON_DRAW_DATABASE

struct ICON_VECTOR_3005
      Glfloat     V1[4];
                  V2[4];
                  V3[4];
                  V4[4];
                  V5[4];
                  V6[4];
                  V7[4];
```

```
void DRAW_ICON_3005(ICON_VECTOR_3005);
{
      glNewList(ICON_NUMBER, GL_COMPILE);
            glBegin(GL_POLYGON)
                  glColor3f(COLOR_1, COLOR_2, COLOR_3);

                  glVertex4fv(V1);
                  glVertex4fv(V2);
                  glVertex4fv(V3);

                  glLineWidth(LINE_WIDTH);

                  glLineStipple(1x3f07);
                  glEnable(GL_LINE_STIPPLE);

                  glColor3f(COLOR_4, COLOR_5, COLOR_6);

                  glVertex4fv(V4);
                  glVertex4fv(V5);
                  glVertex4fv(V6);
                  glVertex4fv(V7);
            glEnd();
      glEndList();

}
```

---

9. DYNAMIC RUN-TIME SYSTEM ARCHITECTURE

The critical part of any complex software design is the architecture. This is what determines the supportability (maintainability) of the end product. In the case of a general purpose interactive graphical support system such as RTG, the critical nature of the architecture is extreme. This is because of the requirement to handle events generated by input devices, e.g., the mouse and keyboard, whenever the user decides to use them. The tree of possible input events is huge when allowing the user to group icons into hierarchies, scale, translate, and rotate icons, and do many other activities, e.g., add text boxes with scaleable fonts. In addition, this tree must be designed to be traversed in many ways, at the whim of the user, to insure being very user friendly.

Now add to these requirements the need to interact with a running simulation. Add again the need to field numerous graphic events from this running simulation and the architecture becomes of paramount importance. The structure of the architecture must support handling these disparate events in a sufficiently fast and efficient manner. Finally, to survive, a new and innovative software system must be adaptive; it must accommodate changes and upgrades easily as more is learned about how a user will want to use it. Thus the structure must be made up of independent modules that are easily understood so that changes in functionality are easily accommodated.

Such a design must start by addressing the design of modules to handle the various disparate event streams that can occur. This is best accommodated by treating these event streams as independent parallel paths. In effect, an event causes a path to be taken to the point where the function specified by the user is completed. Once completed, the path is terminated, and another event stream can be processed with an entirely different path.

While these events are being fielded, the simulation may be running, generating graphical output events and queries for user inputs. Interactions between user input events and simulation events must be carefully considered. Since RTG successfully supports these functions, we will follow the RTG design philosophy for that part of the architecture involving concurrent processing of the simulation and fielding of graphical input events, making the appropriate modifications.

What will be changed is the architecture for handling events. In effect, the design will evolve around a state vector approach that has proved to be the best way to characterize complex nonhomogeneous systems. With this approach, the design is embodied in the separation of the overall system state vector into independent subvectors. These subvectors need only be available to those processes being invoked to field the particular event stream being fielded at the time.

A TYPICAL EVENT HANDLING ARCHITECTURE

Figure 9-1 illustrates the architecture of a typical software system that incorporates event handling. Using this approach, the event handler is at the top of a large loop, commonly known as the *main-loop* in graphics. Approaching the architecture this way, the designer maps out all of the routines that are spawned from the event handler as shown in the figure.

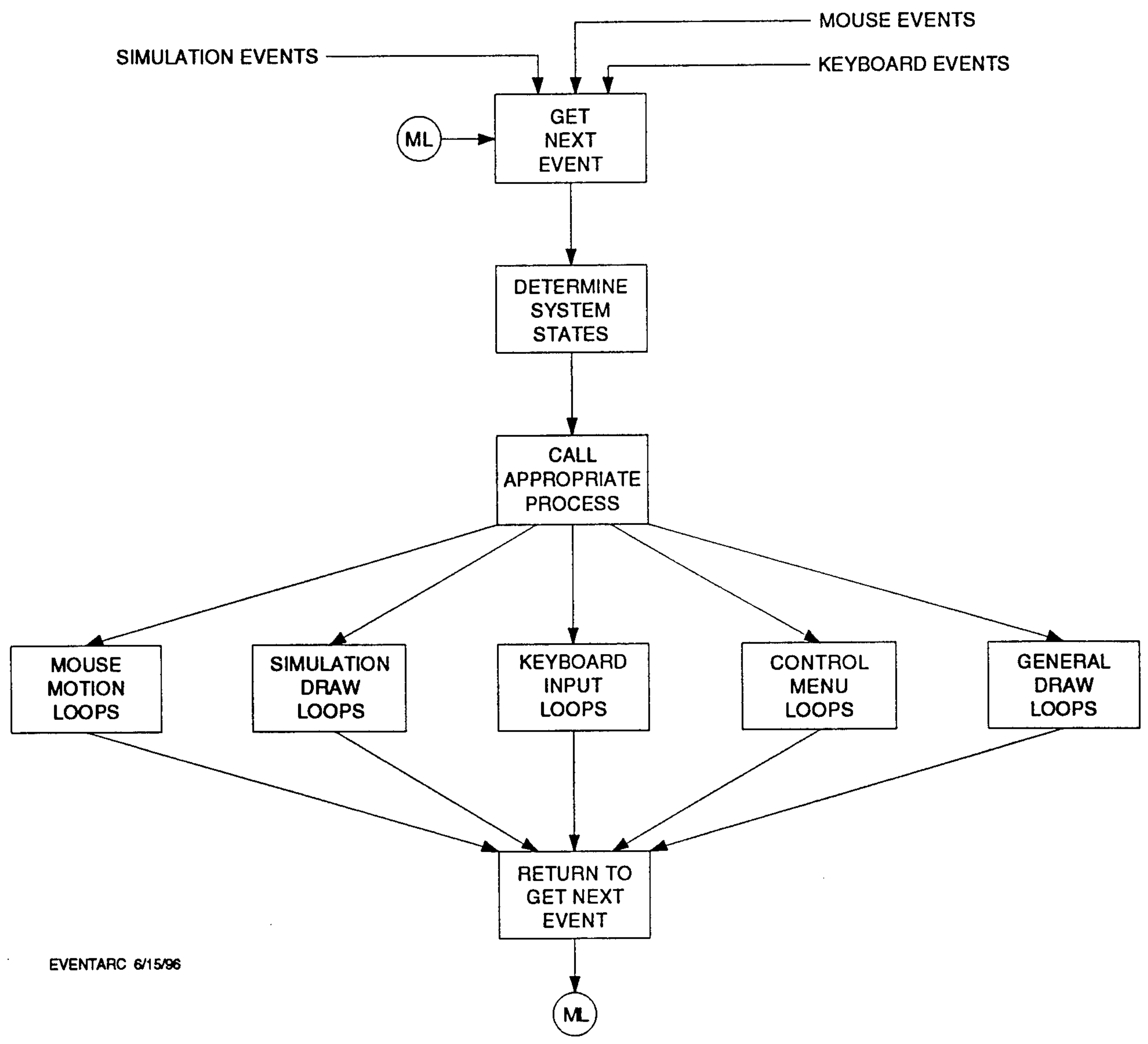

Figure 9-1. Example of control flow architecture implementing a typical *main-loop* event handler.

This architecture is based upon the sequential control flow concepts of a conventional software system. Every part of the architecture fits into a sequential step-by-step process. The sequential nature of the control flow representation inhibits conceptual rationalization of parallel systems. This presents a major barrier to the kind of design problem we want to solve.

In the RTG environment, the simulation can be running in many parallel paths while the graphics facilities are running concurrently. Since we will want to be positioned to take advantage of a parallel processing environment, we don't want to be restricted to a sequentially based architecture.

AN ALTERNATIVE EVENT HANDLING ARCHITECTURE

An alternative architecture is shown in Figure 9-2. Using this approach, the event handler is removed from the center of control. Instead, a top level control process calls the appropriate processes to implement the desired user functions. These processes may call the event handler if and when it is needed, with control returning to the calling process after the event is fielded.

With this architecture, the design evolves from the functional requirements attendant to supporting a friendly graphical user interface. In RTG, there are a large number of functions desired by the user. In general, these functions can be organized into a tree structure. Selection of the functional path at the upper levels of the tree can be accomplished using a corresponding hierarchical menu structure. At the bottom of the path, answers to specific prompts or actions taken by the user must be fielded. Query and response to these actions is generally supported by specific modules. Any of these modules can in turn call the desired event handler utility with direct control for tight loops, e.g., the mouse box-select loop or the drag-icon loop.

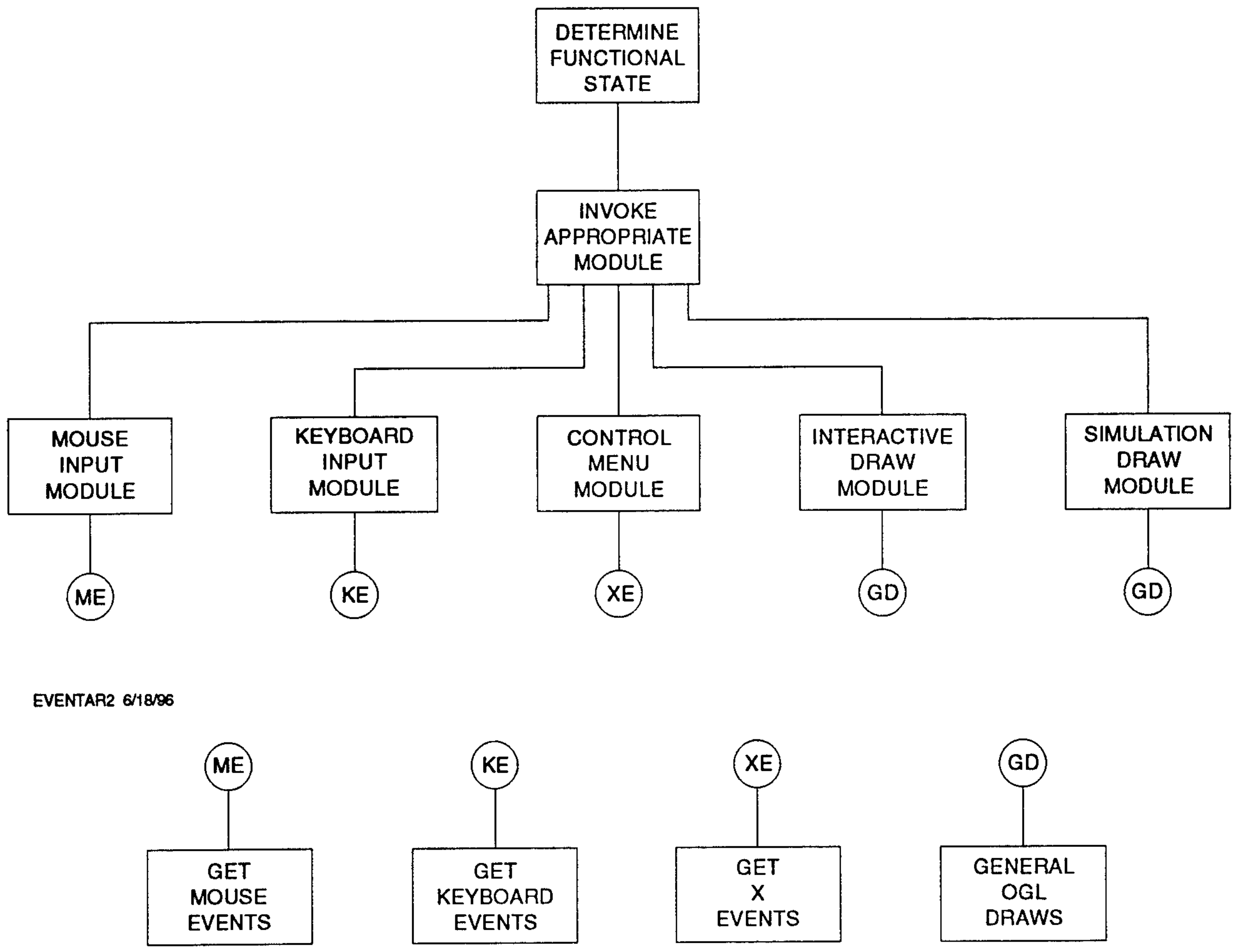

Figure 9-2. Example of a functional oriented architecture that is independent of control flow.

With this alternative architecture, the event handler is subordinate to any routine that wants to call it. This distributes control to where it's needed as opposed to having the event handler design dominate the architecture. In addition, it provides for the design of high-speed loops to support the handling of multiple mouse events, such as dragging icons or selecting groups of icons with a box select. To understand how this can be implemented, we have taken slices of critical user scenarios and analyzed what the user desires functionally. We then implemented the corresponding slices of the system architecture in the demonstration facility to support these functions.

This type of architecture places emphasis upon the use of hierarchies of utilities. For example, at the bottom of Figure 9-2 is a set of lower level utilities. These utilities may invoke yet lower level utilities. They are invoked by higher level modules that themselves may be utilities. This layering of utilities is common in the architectural designs produced by PSI for interactive graphical facilities. Many similar X-Windows utilities have already been developed for the GSS monitor. These include the hierarchical menu selection utility, the list selection utility, various button boxes, and prompted input facilities.

IMPLEMENTING BACKGROUND OVERLAYS

As in the current RTG system, up to eight background overlays will be supported. This will be implemented using a color index database similar to RTG. Users who want to tailor special background overlays will be able to modify the color index database. Background overlays will be assigned to separate bitplanes in the framebuffer.

Bitplane Assignments, Color-Index Mode, And Color Mapping

As in the current RTG system, color-index mode is the best way to separate background overlays. In this mode, the user can assign one or more bitplanes to each background overlay. In a system with 24 bitplanes for color and double buffering, 12 bitplanes will be available for each color buffer. The foreground will use 4 of these to provide 16 colors. As an example, these could be black, white, red, green, blue, and 11 others. Then 8 bitplanes will be left for background overlays.

Users will be able to assign these 8 bitplanes to background overlays as desired. For example, if six background overlays are to be used, then two of these could have 2 bitplanes each, and the remaining four would have 1 each. The user will be able to select the colors for the background overlays with the exception that black (no color) must always be one of the selections. Thus, if 1 bitplane is used, its two states will indicate whether on not the color is on. If 2 bitplanes are used, the four states can be black and three colors.

In color-index mode, 32 colors should be sufficient. These will generally include the primary colors made up by the RGB numbers being zero or maximum. The resulting colors are black, white, red, blue, green, yellow, cyan, and magenta. The colors will be specified in a *color-index to RGB* mapping file in the RTG system directory.

User-Written Object Libraries

Background overlays are written by users, in VSE, C, FORTRAN, or any other language that can be compiled into an object library and called from VSE, GSS, or C. They are usually complex scenes that remain stationary. Examples are terrain contours, foliage, towns, roadways, waterways, grids, and other map items. They are typically written from special user databases, such as Defense Mapping Agency (DMA) databases. These database layers can be toggled on or off while the simulation is running by choosing background overlay selections in the RTG control window, or by commands from the simulation.

When a background overlay is toggled on, it will be drawn on the display screen by a call from GSS to the background overlay routine (written by the user) to do the drawing. The background scene will remain unchanged until the background overlay is changed or toggled off. Changes will occur when panning or zooming occurs. Thus, many changes may be made to the foreground, e.g., changes to icons, lines, instruments, legends, user menus, control window, etc. while the background remains unchanged with no need to be redrawn. Separating background scenes from foreground draws helps to speed the graphics response by considerable multipliers.

It is up to the user to learn the graphics facilities of the particular platform being used, and to understand the information in the resources shared with RTG.

IMPLEMENTING TEXT

A set of fonts will be provided for supporting text boxes and labels as well as for the system control boxes. These will be built as database files that can be changed by an advanced user. Fonts will be scaleable so they grow larger or smaller with the other graphics objects as the user zooms in or out or reshapes the RTG window.

Text labels and text boxes must be defined prior to running a simulation. They will be created in the drawing board edit mode. Text in menus can be specified using a special version of the GSS resource as is done in the current version of RTG-X used for the VSE and GSS monitors.

10. DESIGN AND IMPLEMENTATION OF THE DEMONSTRATION

Most of the effort expended during the final reporting period was focused upon implementing and testing various critical elements of the design. These elements were forged together into a graphics facility that provides a demonstration of what we will be doing in Phase II, as well as the research and design performed in Phase I. This graphics demonstration facility contains a sufficient subset of the elements required in the final system to provide a solid foundation for the Phase II architecture and the corresponding development effort. The functional elements contained in the demonstration facility are listed below.

- Insert icons
- Move (drag) icons into their desired positions
- Select a group of icons
- Represent (cover) a selected group of icons with a different (higher level) icon
- Move the higher level icon from one position to another
- Lift the higher level icon cover to expose the group of icons underneath in their new positions

The architecture of the demonstration facility follows that of the final system architecture. It utilizes all of the graphics facilities that will be needed to complete the final system. The most important set of graphics facilities contained in the demonstration are enumerated below.

- RTG Graphics control window
- Functional selection control menus
- Mouse event handler
- Keyboard event handler
- Hierarchical icon databases
- Icon pick and select modules
- Open GL draw lists and calls
- X-Windows widgets and calls
- Scaleable fonts

A demonstration facility containing the results of the research and development of the new facilities described in the SBIR Phase I proposal have been completed. Demonstrations were given using this facility, as well as the existing Run-Time Graphics (RTG) facility, at the GSS User's Group Conference on 11 and 12 July, 1996 for the interested parties at CECOM, DISA/JIEO, and other Army organizations and contractors. This new facility demonstrates that the various research efforts performed by PSI have resolved all of the known outstanding design issues. Given the knowledge base existing in the current RTG system, and the research and development completed during this SBIR Phase I effort, it remains to implement the details of the design in accordance with the specified architecture.

11. SUMMARY

We have completed the design of the hierarchical icon facility as it will fit into an overall simulation run-time graphics system. We have investigated the required hierarchy of RTG states, the use of text boxes, the use of menus, and the overall control window facilities required to support a complete graphical simulation facility. We have completed the database design to support moves of hierarchical icons from one spot in the hierarchy to another.

We have also built and tested a substantial part of the mouse pick and select modules. This has been accomplished through extensive testing of drawing responses to fast mouse motion when creating select boxes or dragging icons. We have designed the overall user interface in terms of states and transitions from state to state. We have implemented a scaleable font alphabet to support zooming, rotating and stretching of icons whose text labels remain in proportion and position with the icons. We have completed the details of the design that supports handling input events from the simulation as well as from the user at the interactive workstation.

Very little of the above efforts could have been performed in the period of performance of this SBIR Phase I contract had PSI not had substantial experience in the development of graphical interfaces to support communication network simulation in a complex and sophisticated customer environment. Having built and supported upgrades to the existing RTG system over the past nine years, PSI has gained a wealth of knowledge about what the users want, how to achieve platform independence in a sophisticated graphics environment, and how to design the software architectures so that the they are flexible to accommodate change as new functions and features are desired.

The use of Run-Time Graphics, as opposed to Model Development Graphics is finally being appreciated by a growing number of PSI's clients. Most modelers think of graphics as an output facility, to show what has happened in the simulation. Some more advanced modelers use it as a model development tool - one that requires compilation after the model is built.

Using a totally new technological approach, GSS users have the ability to construct and modify their models while the simulation is running. This allows one to observe the changing results, and make informed decisions regarding how to restructure the model based upon the unfolding scenario. This dynamic operating environment eliminates the need to recompile the model before running another simulation, or to review results graphically after the simulation has ended.

By changing the network interactively, dynamically while the simulation is in progress, one obtains immediate answers and feedback regarding network designs. This allows analysts to construct experiments on the fly. It provides a modeling and simulation technology that directly supports the requirements for Distributed Interactive Simulation (DIS) experiments right now. This interactive graphical DIS facility was demonstrated at the GSS User's Group Conference in July for command and control in Tactical Operations Centers (TOCs) as well as for communication equipment deployment. This facility has put the Army ahead of other organizations in modeling and simulation of communication and control systems.

By incorporating the hierarchical icon capability, it will be extremely easy to construct and vary complex networks and their environments and get simulation results graphically. This will help analysts to build and control the very large scenarios needed to construct a total communications picture of the battlefield. The hardware horsepower is catching up to support these experiments. This effort will put the put the GSS-RTG tools back ahead of the hardware. This is precisely what the experienced Army modelers now need and want.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE 8-15-96	3. REPORT TYPE AND DATES COVERED FINAL

4. TITLE AND SUBTITLE NETWORK SIMULATION OF TECHNICAL ARCHITECTURE .	5. FUNDING NUMBERS Contract DAAKF11-96-P-0347
6. AUTHOR(S) William C. Cave, Robert E. Wassmer, Ken Irvine	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Prediction Systems, Inc. 309 Morris Avenue Suite G Spring Lake, NJ 07762	8. PERFORMING ORGANIZATION REPORT NUMBER PSI - 96001
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ARMY ATLANTA CONTRACTING CENTER TECHNOLOGY CONTRACTING SECTION Bldg 130 Anderson Way AAFLG PRCT Fort McPherson, GA 30330-6000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES NONE	
12a. DISTRIBUTION/AVAILABILITY STATEMENT UNLIMITED	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report describes the SBIR Phase I design of an interactive graphics facility for network modeling to simulate battle command technical architectures to guide the definition, design, and development of Army battle command systems. This facility supports modeling of network architectures of interoperable telecommunications and information systems among different command levels and their user systems. It is based on technology currently used by the Army to model a wide array of systems, including various tactical radios and their environments, satellite systems, switched systems, including circuit, packet, and ATM, high capacity local area networks, PCS, and a broad range of voice and data subscribers. Models of various weapons and command and control systems also exist, including versions used in Distributed Interactive Simulation (DIS) experiments.

Phase I explored and demonstrated significant upgrades to advanced run-time graphics that allow users to create and modify architectures and scenarios - while simulations are running - in real-time. The new technology facilitates the ease with which users can interactively manipulate and control complex hierarchies of organizations and equipment using iconic models, and watch responses to system dynamics under various stress conditions using a braod selection of graphical depictions and visual instrumentation.

14. SUBJECT TERMS DISCRETE EVENT SIMULATION MODEL DEVELOPMENT GRAPHICS HIERARCHICAL RUN-TIME GRAPHICS COMMUNICATIONS MODELS	15. NUMBER OF PAGES 53
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

NSN 7540-01-280-5500

REF 9

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102